Due: 03.02.2022 (23:59 CET) **Deep Learning** F. Hutter & A. Valada
Points: 16 + 4 bonus 13^{th} Assignment WS 2021/2022

To get access to this week's code use the following link: https://classroom.github.com/a/teYe7IlW

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.

In this exercise, we will look at methods to get uncertainties with Neural Networks. More specifically, we will

- Implement Bayesian Linear Regression
- Get uncertainty estimates using DNGO
- Get uncertainty estimates using Ensembles
- Get uncertainty estimates using PFNs (bonus)

1. Bayesian Linear Regression (BLR)

1) [3 points] BLR Implementation

In this exercise, we are going to implement Bayesian Linear Regression where we assume a Gaussian prior having 0 mean and covariance Σ_p. Also, the bias weights are not learned and assumed to be 0.

Note: The slides for this part are based on Chapter 2 of the GPML book, thus, everytime we refer to equation numbers, you can also find these in the book.

Todo: complete the function `linreg_bayes` in `lib/blr.py`. You should implement the Gaussian posterior as stated in eqn. 2.8.

Note: The BLR class also takes mean of the prior μ_p as argument. You may ignore it (eqn. 2.8 assumes $\mu_p = 0$), i.e. you do not need to implement the case $\mu_p \neq 0$.

You can check the correctness of your implementation by running `tests/test_linreg_bayes.py`.

Todo: complete the function `posterior_predictive` in `lib/blr.py`. Here you should implement the posterior predictive distribution according to eqn. 2.9.

Note: Eqn. 2.9 defines the predictive distribution for a single data point x. Your implementation should work for a batch of datapoints.

You can check the correctness of your implementation by running `tests/test_post_predictive.py`.

2) [2 points] **Plot Samples from Prior/Posterior** We can test whether this works as expected by running the coded Bayesian Linear Regression method on the 1d dataset. We will sample from the prior and posterior distribution that we get from fitting the Bayesian Linear Regression (`linreg_bayes`) and plot the different linear models corresponding to each sampled weight.

Todo: Complete the function `plot_bayes_linear_regression` in `lib/plot.py` to plot the samples from the prior distribution and additionally (on top) the samples from the posterior to see the differences. Make sure to plot the models sampled from the prior/posterior in two different colors. Also include the prior mean (μ_p) and posterior mean (μ_{post}) in the plot. Finally, use varying alpha (opacity) values to indicate the relative likelihood of models ($\sim$ probability density), i.e. use a higher transparancy (lower alpha $\sim$ lighter color) for less likely models.

Note: Recall that we assume the bias weights to be 0 in this part (but not in the DNGO part later)

3) [2 points] **Plot Contour & Answer Question** Now we perform BLR for a 2d dataset and generate contour plots of the pdf of the prior and posterior distributions.

Todo: Complete `plot_contour` in `lib/plot.py`. Your task is to use the data and compute the Gaussian posterior with `linreg_bayes`. Furthermore, you should instantiate the multivariate Gaussian distributions for the prior and the posterior, using `scipy.stats.multivariate_normal`.

After completing the functions, run `run_lin_regression.py` to see the plots.

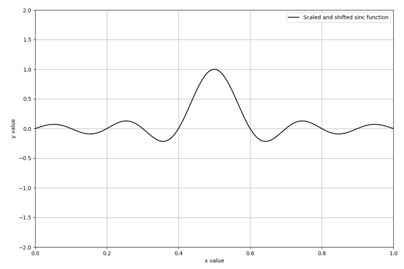
Answer in pdf: How do the prior and posterior distributions differ? Why?

Hint: You should explain the difference between the prior and posterior distributions, and *not* the difference between the 1D and 2D plot.

2. Uncertainties using DNGO

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

In the remainder of this 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 an NN to completion and then takes the learned features from the last hidden layer and performs BLR (`linreg_bayes`) on it.

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

Hint: Reuse the functions you have implemented in the first exercise.

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

You can check if the implemented functions are correct by using `tests/test_dngo.py`

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) [2 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 you the predictions and ground truth values for the grid and train values. The second plot then shows the uncertainty at each point for different uncertainty bands.

Answer the following questions (in 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)?

3. **Uncertainties using Ensembles**

1) [2 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.

You can test the correctness of your implementation by using `tests/test_ensemble.py`

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) [2 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 in pdf: How do these results compare to those of DNGO? Is this comparison fair?

4. [2 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 tranformer 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 (in 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 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 03.02.2022 (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.