Exam project for PML 2022/2023

The exam project consists of parts A and B, reflecting different topics from the course. You are expected to work on the project in groups of 2-4 students. To avoid any misunderstandings, please read the following sections about requirements and practicalities carefully before you begin.

Technical requirements and practicalities

Report length The project should be documented in a brief report (one report per group), which will determine your final grade in the course. The report should consist of maximum 6 pages in a reasonable font-size, including figures. You are allowed to include additional text, results and figures as appendices (these do not count to your total page number).

Report style We recommend using this template for your report: https://da.overleaf.com/latex/templates/style-and-template-for-preprints-arxiv-bio-arxiv/fxsnsrzpnvwc.

Should I include code in my report? If you want to highlight a few particularly relevant lines of code, that's fine - but in general we do not expect code in the report. You can optionally include it in an appendix, but the most useful approach is to provide a link to a relevant github repository.

Contribution from group members If members of the group contributed unequally to the different parts of the project, you should state this in the report, by specifying for each part the contribution percentage from the different members. If nothing is stated, we will assume that all members contributed equally.

Handing in The reports should be handed in as a PDF through Digital Exam (eksamen.ku.dk), which will become available over the next week.

Questions The next two weeks, the exercise sessions (Thursday 13:00-16:00) will be dedicated to the projects. Richard will be there as usual, but in addition, Thomas, Oswin and Wouter will join from 13:00-14:30 on both Thursdays to avoid that Richard is overloaded with questions. We prefer that you save your questions for the in-class sessions, but if something urgent comes up, you are welcome to start a thread on the Absalon Discussion Board, and we'll try to answer it there. Please do not send us emails with questions about the project (since we then end up answering the same questions many times).

A Density modeling

During one of the exercise sessions, we implemented a basic variational autoencoder (VAE) on MNIST data. In the first part of the project we will consider extensions and alternative models to fit the same data, and compare them to the basic VAE that you did in class. Note that a sketch of a solution to the VAE exercise from class is available in Absalon under `Files->Solutions`.

A.1 Implement a convolutional VAE

The first task is to extend the original model to use convolutional layers rather than linear layers. Compare the quality of the model to the original. Are the models significantly different?

Deliverables:

- Briefly describe how the VAE was implemented, and how parameters were estimated.
- Describe the performance of the model compared to the original. You can do this visually, based on the quality of generated images, but try to also come up with a way to do this quantitatively.

A.2 Alternative models

Now implement one or more alternative density models on the same data. Here are some examples of model choices (some more ambitious than others):

Mixture model A discrete latent variable model.

Probabilistic PCA A simpler continuous latent variable model than the basic VAE.

Bayesian VAE A model where you consider distributions of the weights in the neural network of your decoder. You can estimate such a model using variational inference on both the latent variable and the weights.

Diffusion model In recent years, diffusion models have established themselves as the state-of-the-art in generative models for images.
Note: published diffusion models often have many ingredients, including U-NET architectures and attention layers - to avoid getting lost in implementation details, we recommend starting with a simple model, e.g. using just a simple convolution network for the denoising network.

Deliverables:

- Argue for your choice of models and describe essential properties and advantages/disadvantages of these models (e.g. tractability of likelihood)
- For each model, assess whether the model is better or worse than the original VAE (preferrably both by visual inspection and by quantifying the goodness of fit).

B Function fitting

In this part, you will use Gaussian Processes for function optimization. Use Pyro to implement a fully Bayesian GP and then use it to find a local minimum of the function

$$f(x) = \sin(20x) + 2\cos(14x) - 2\sin(6x)$$

on the interval $x \in [-1, 1]$

B.1 Fitting a GP with Pyro

First follow the Pyro GP tutorial on GP Regression and afterwards read the Pyro GP Documentation. Your task is to implement a full Bayesian GP modeling approach where you first use MCMC sampling to obtain the posterior hyperparameters θ of the GP and then use these samples to provide mean and variance estimates for observations at new points. As dataset $\mathcal{D}$, use the five input-label pairs (x_i, y_i) with $x_i = -1, -1/2, 0, 1/2, 1$ and y-values $y_i = f(x_i)$ using the function above. Follow the steps below:

1. Use NUTS to sample from the posterior $p(\theta|\mathcal{D})$.
2. Check the quality of the MCMC sampling using diagnostics (hint: use Arviz). Use the diagnostics to choose the hyperparameters of the sampling (such as the number of warmup samples).
3. Use the obtained MCMC samples from the posterior to obtain estimates of mean $m(x^*)$ and variance $v(x^*)$ of $p(y^*|x^*, \mathcal{D})$ at a point $x^* \in [-1, 1]$

Deliverables:

- A scatter plot on log-log-scale of $N = 500$ samples from $p(\theta|\mathcal{D})$.
- An analysis of your obtained sample quality.
- A plot visualizing $p(f^*|x^*, \mathcal{D})$ for $x^* \in [-1, 1]$. For this, overlay a plot $f(x)$, a scatter plot of $\mathcal{D}$, and plots of $m(x^*)$ as well as $m(x^*) \pm 2\sqrt{v(x^*)}$. Add proper labels and axis descriptions.

For your deliverables, use the following GP parameters:

- Kernel: Gaussian RBM kernel with parameters `lengthscale` σ_l^2 (kernel width) and `variance` σ_s^2.
- Priors: use a `LogNormal` prior:

$$\sigma_l^2 \sim \text{LogNormal}(-1, 1)$$
$$\sigma_v^2 \sim \text{LogNormal}(0, 2)$$

- Fixed GP noise variance 10^{-4}.

```
Data: Initial dataset $\mathcal{D}$. Candidate set $X^* = \{x_1^*, \dots x_\ell^*\}$, number of
      iterations $T$
for $k = 1, \dots, T$ do
    Sample $f^* \sim p(f^*|X^*, \mathcal{D})$;
    $p = \arg\min_i f_i^*$;
    Add $(x_p^*, f(x_p^*))$ to the dataset $\mathcal{D}$
```

Algorithm 1: Bayesian Optimization Algorithm

B.2 Bayesian Optimization

Now, use the code from B.1 to build a Bayesian Optimization loop to find a local optimum of f. For this, implement Algorithm 1. In each iteration, sample a function observed on X^* from the posterior predictive of the GP using the dataset. Then you locate the minimum of the sampled function on X^* and evaluate f at this position. Finally, add the new position/value pair to $\mathcal{D}$. Repeat until a target number of iterations is reached. For the GP use the same prior parameters as in the previous task and you should also ensure sufficient convergence of the MCMC chain. For X^* use 200 evenly spaced points in $[-1, 1]$

Deliverables:

1. A plot of f, the sampled f^*, confidence intervals $m(x^*) \pm 2\sqrt{v(x^*)}$ for $x^* \in X^*$, at iterations $k = 0, 5, 10$. In the plots, also mark the location of the added pair $(x_p^*, f(x_p^*))$. Add proper labels and axis descriptions.

2. Run the algorithm several times and note your observations. Does it always find the global optimum? If not, what shortcomings do you observe?

3. Can you find ways to fix the shortcomings? e.g., are there better choices for kernel or prior or is there a better way to select new points? Investigate one idea.

Hint 1: To obtain a sample f^* only a single sample from $p(\theta|\mathcal{D})$ is needed.
Hint 2: For numerical stability, f^* should not be sampled noise-free.
Hint 3: if it appears that Pyro ignores your setting of a kernel parameter, consider cleaning up old variables with `pyro.clear_param_store()`.