Task 2 - Bayesian Neural Nets

1. Task description

In this task, you will implement a *Bayesian Neural Network* that yields well-calibrated predictions.

Task Background

Bayesian Neural Networks

Neural networks have enjoyed many successes in recent years across a variety of domains. Many practitioners claim that to bring neural networks into more risky domains such as healthcare and self-driving vehicles, neural networks must be able to estimate their prediction's uncertainty. An approach for doing this involves using *Bayesian Neural Networks (BNNs)*. BNNs combine Bayesian principles of model averaging with the black-box approach of deep learning. As a result, BNN probability outputs are better calibrated in comparison to traditional neural networks.

During the lectures, you have already seen one way to implement BNNs (Graves, 2011). However, that method requires the closed-form solution of a KL-divergence between the weight prior and corresponding variational posterior. In this task, we focus on the full *Bayes by backprop* approach (Blundell et al., 2015). Bayes by backprop yields unbiased gradient estimates and supports a large set of priors and variational posteriors. While the algorithm is very similar to what you have seen in the lecture, we still encourage you to carefully read the paper (Blundell et al., 2015) before solving this task.

Calibration

In what follows, we utilize the notation and terminology of Guo et al. (2015).

We focus on supervised learning problems with features $X \in \mathcal{X}$ and labels $Y \in \mathcal{Y}$, where both X and Y are random variables. In this task, $\mathcal{X} = \mathbb{R}^{784}$ are flattened images and $\mathcal{Y} = \{0, \ldots, 9\}$ is a discrete label space. Given some $X \in \mathcal{X}$, a neural network $h(X) = (\widehat{Y}, \widehat{P})$ outputs a tuple consisting of a label $\widehat{Y} \in \mathcal{Y}$ as well a confidence $\widehat{P} \in [0, 1]$.

We say that a model is *perfectly calibrated* if its confidence matches its performance, that is,

$$\mathbb{P}(\widehat{Y} = Y \mid \widehat{P} = p) = p, \quad \forall\, p \in [0, 1],$$

where the randomness is jointly over X and Y.

To understand why uncertainty calibration for predictive models is important, consider the task of relieving doctors in medical diagnosis. Suppose that real doctors are correct in 98% of all cases while your model's diagnostic accuracy is 95%. In itself, your model is not a useful tool as the doctors do not want to incur an additional 3 percentage points of incorrect diagnoses.

Now imagine that your model is well-calibrated. That is, it can well estimate the probability of its predictions being correct. For 50% of the patients, your model estimates this probability at 99% and is indeed correct for 99% of those diagnoses. The doctors will happily entrust those patients to your algorithm. The other 50% of patients are harder to diagnose, so your model estimates its accuracy on the harder cases to be only 91%, which also coincides with the model's actual performance. The doctors will diagnose those harder cases themselves. While your model is overall less accurate than the doctors, thanks to its calibrated uncertainty estimates, we managed to increase the overall diagnostic accuracy (to 98.5%) and make the doctors' jobs about 50% easier.

Problem Setup

Your Task

Your task is to implement a Bayesian Neural Network for multi-class classification and train it using the Bayes by backprop algorithm as introduced in Section 3 of Blundell et al. (2015). Furthermore, you will need to determine and implement suitable weight priors so that your BNN produces well-calibrated uncertainty estimates. The goal is to evaluate how well the BNN estimates the uncertainty of its predictions. Remember that the remarkable property of BNNs is their ability to "measure" uncertainty. To provide a solid intuition for this behavior, we use a special test set so that your BNN's predictions experience strong epistemic and aleatoric uncertainty .

We provide you with a solution template `solution.py` in the handout. Your concrete tasks are as follows:

- Implement the class `BayesianLayer` that represents the Bayesian equivalent of an affine neural network layer.
- Implement the class `BayesNet` that represents a full BNN.
- Determine and implement suitable weight priors and variational posteriors.
- Implement the class `Model` to perform Bayes by backprop and use the resulting network for inference.

The template contains an abstract class `ParameterDistribution` that models arbitrary distributions over weights (both priors and posteriors). We already provide the structure for univariate and diagonal multivariate Gaussians in the classes `UnivariateGaussian` and `MultivariateDiagonalGaussian` respectively. You should implement those two classes, but also play around with prior parameters or more complex prior distributions that yield better uncertainty estimates.

To highlight how BNNs are better calibrated than ordinary neural networks, we provide a non-Bayesian neural network as a reference. If you set `use_densenet` in the `Model` class to `True`, your BNN is replaced by an ordinary neural network implemented in the class `DenseNet`. Your BNN should have a much lower calibration error than the ordinary neural network.

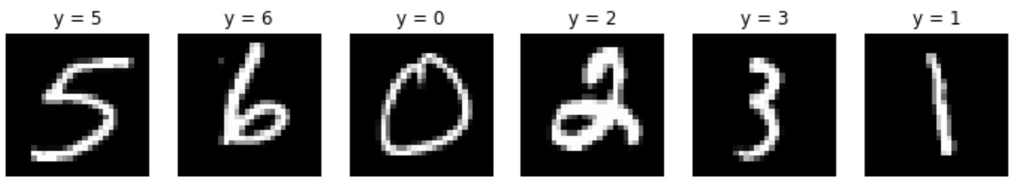
Note that the template in `solution.py` uses *PyTorch* as its neural network library. If you are new to PyTorch, you might want to check out the PyTorch quickstart guide. While you are technically free to use a different library, you will have to re-implement a significant amount of code. We can thus only provide support for PyTorch-related issues.

Dataset

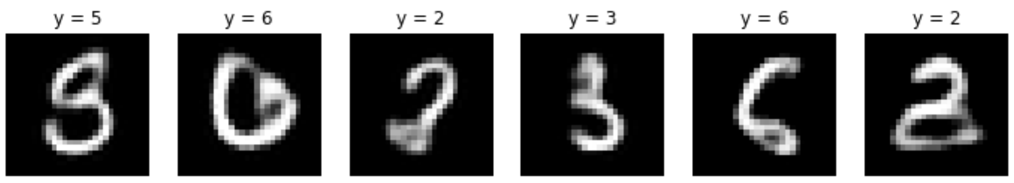
One of the most common datasets in the field of machine learning is the MNIST dataset (LeCun et al., 1998). MNIST is a large database of handwritten digits which is commonly used for training various image processing systems. The MNIST dataset contains 60000 training images, each consisting of 28x28 grayscale pixels with a label in the range $\{0, \ldots, 9\}$.

The training set consists of the original 60000 MNIST training images. Since we want to restrict ourselves to simple feedforward neural networks (in contrast to convolutional ones), we reshape all images (in both training and test sets) into 784-dimensional vectors for you. For the test set, we use a modified version of MNIST. The test images exhibit varying degrees of ambiguity and are further rotated by random angles. Ambiguity introduces aleatoric uncertainty since a true image label might not be uniquely identifiable. The rotations introduce epistemic uncertainty since your network only observes non-rotated images during training.

The following are example training set images:



The following are example test set images:



As in Task 1, the test dataset is split into two parts. This split is not revealed to you. We calculate the PUBLIC metrics and score only on the first part of the data while we use all test samples for the PRIVATE score. You can use the provided script (see below for the workflow) to generate the public metrics and cost. However, you can observe the PRIVATE score only once you submit your solution to the server. You pass the task if your solution's PUBLIC ECE and accuracy satisfy the constraints described below. The PRIVATE score is *irrelevant for grading* and only counts for the leaderboard. We will invite the highest-scoring teams to present their approaches in a tutorial session after this task's hand-in deadline.

Metrics

Remember that for predictions $\widehat{Y}$ with confidences $\widehat{P}$, your model is well-calibrated if $\mathbb{P}(\widehat{Y} = Y \mid \widehat{P} = p) \approx p, \forall\, p \in [0, 1]$. An obvious error measure is the expected absolute difference between the true accuracy $\mathbb{P}(\widehat{Y} = Y \mid \widehat{P} = p)$ and the confidence p, that is,

$$\mathbb{E}_{p \sim \widehat{P}}\left[\left|\mathbb{P}(\widehat{Y} = Y \mid \widehat{P} = p) - p\right|\right].$$

This criterion is called the *Expected Calibration Error (ECE).*

However, we do not know the true accuracy $\mathbb{P}(\widehat{Y} = Y \mid \widehat{P} = p)$ in practice. In this task, we approximate the true accuracy via quantization over M intervals, leading to the *empirical accuracy*. For $m \in [M]$, let B_m denote the set of indices whose predicted confidence $\widehat{p}_i$ falls into the interval $I_m = (\frac{m-1}{M}, \frac{m}{M}]$. Then, the empirical accuracy is

$$\mathrm{acc}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} 1_{\widehat{y}_i = y_i}.$$

Similarly, we define the *empirical confidence* as

$$\mathrm{conf}(B_m) = \frac{1}{|B_m|} \sum_{i \in B_m} \widehat{p}_i.$$

This leads to the natural definition of the *empirical ECE* as

$$\widehat{\mathrm{ECE}} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\mathrm{acc}(B_m) - \mathrm{conf}(B_m)|.$$

You can find our implementation of the empirical ECE in `util.py`. For more details, we refer you to Guo et al. (2015).

We use the empirical ECE with $M = 30$ bins to measure your BNN's calibration. Since we also want your BNN to predict well on average, we further measure your model's accuracy. You pass this task if you match our baseline *both* in terms of empirical ECE and accuracy. Concretely, you pass if your model's metrics on the PUBLIC test set satisfy

ECE	Accuracy
<= 0.05	>= 0.65

Note that, while 65% accuracy seems low, the large amounts of uncertainty in the test set will make it difficult to reach significantly higher accuracy levels.

Lastly, we use a compound score defined as

$$\text{score} = \text{accuracy} + 3 * \left(\frac{1}{2} - \widehat{\text{ECE}}\right).$$

This score is *irrelevant for passing* and only determines your position on the leaderboard. The checker will reveal your compound score on the PUBLIC test set while the leaderboard on the server uses the PRIVATE compound score.

Submission Workflow

1. Install and start Docker. Understanding how Docker works and how to use it is beyond the scope of the project. Nevertheless, if you are interested, you could read about Docker's use cases.
2. Download handout
3. The handout contains the solution template `solution.py`. You need to implement all sections marked with `# TODO: ...`. Please make sure that your implementation preserves the names and signatures of the `run_solution` method, as well as all methods in the classes `Model`, `BayesianLayer`, and `BayesNet` since the evaluation script relies on them. However, you are free to introduce new methods and attributes. Note: The `main()` method in the solution template is *for illustrative purposes only* and *completely ignored* by the checker!
4. You should use Python 3.8. You are free to use any other libraries that are not already imported in the solution template. Important: please make sure that you list all additional packages together with their versions in the `requirements.txt` file provided in the handout.
5. Once you have implemented your solution, run the checker in Docker:
   - On Linux, run `bash runner.sh`. In some cases, you might need to enable Docker for your user if you see a Docker permission denied error.
   - On MacOS, run `bash runner.sh`. Docker might by default restrict how much memory your solution may use. Running over the memory limit will result in docker writing "Killed" to the terminal. If you encounter out-of-memory issues you can increase the limits as described in the Docker Desktop for Mac user manual. We do not support ARM-based M1 MacBooks yet. If you own such a device, you can resort to the Euler cluster. Please follow the guide specified by *euler-guide.html* in the handout.
   - On Windows, open a PowerShell, change the directory to the handout folder, and run `docker build --tag task2 .; docker run --rm -v "$(pwd):/results" task2`.
6. If the checker fails, it will display an appropriate error message. If the checker runs successfully, it will show you your PUBLIC test accuracy and ECE, tell you whether your solution passes this task, and generate a `results_check.byte` file. The `results_check.byte` file constitutes your submission and needs to be uploaded to the project server along with your code and text description to pass this task.
7. You pass this task if, evaluated on the PUBLIC test set, both your solution's ECE and accuracy satisfy the constraints described in the *Metrics* section. The compound score on the PRIVATE test dataset determines your position on the leaderboard but it does *not* determine whether you pass/fail this task.
8. We limit submissions to the server to 18 per team.

Extended Evaluation

This part of the task is optional but highly encouraged. Once your solution passes the checks, set the global variable `EXTENDED_EVALUATION` in the solution script to `True`. Running the checker again will then create PDF outputs with plots of the image examples within the MNIST data set that your trained network is most confident and least confident about, respectively. By visual inspection, you should be able to notice which features make an image easier or harder to classify and this should correlate with the uncertainty, as visible in the figures. The extended evaluation also shows your predictions on a small subset of the test images. If your BNN is well-calibrated, you should see how it correctly identifies ambiguity.

The extended evaluation will also create the same figures for the FashionMNIST data set. Note that on this data set, the model has never seen any training data, so it should not be able to classify any image better than chance. However, it is still more certain for some images than for others. Observe which features characterize those images and how similar or dissimilar they are to the MNIST images. These experiments highlight a caveat with Bayesian deep learning methods: Their uncertainties might be well-calibrated under mild distribution shifts (ambiguous and rotated MNIST), but can still be unreasonable for strong distribution shifts (FashionMNIST).

Grading

When handing in the task, you need to select which of your submissions will get graded and provide a short description of your approach. This has to be done **individually by each member** of the team. Your submisssion is graded as either **pass** or **fail**. A complete submission typically consists of the following **three components**:

- **Submission file**: The `results_check.byte` file generated by the `runner.sh` script which tries to execute your code and

checks whether it fulfills the requirements of the task.
- Your **code** in form of a .py or .zip file. The source code must be runnable and able to reproduce your uploaded `results_check.byte` file.
- A **description** of your approach that is consistent with your code. If you do not hand in a description of your approach, you may obtain zero points regardless of how well your submission performs.

To pass the task, your submission needs to be complete and **outperform the baseline** in terms of PUBLIC score. Some tasks only have a single score on which you have to improve upon the baseline. Other tasks have a PUBLIC and PRIVATE score. The PRIVATE score determines your position on the server leaderboard but is irrelevant for grading.

Make sure that you properly hand in the task, otherwise you may obtain zero points for this task. If you successfully completed the hand-in, you should see the respective task on the overview page shaded in green.

Frequently asked questions

Which programming language am I supposed to use? What tools am I allowed to use?

You should implement your solutions in Python 3.8. You can use any publicly available code, but you should specify the source as a comment in your code.

Am I allowed to use methods that were not taught in the class?

Yes. Nevertheless, the baselines were designed to be solvable based on the material taught in the class up to the second week of each task.

In what format should I submit the code?

If you changed only the solution file, you can submit it alone. If you changed other files too, then you should submit all changed files in a zip of size max. 1 MB. You can assume that all files from the handout that you have not changed will be available to your code.

Will you check / run my code?

We will check your code and compare it with other submissions. If necessary, we will also run your code. Please make sure that your code is runnable and your results are reproducible (fix the random seeds, etc.). Provide a readme if necessary.

Should I include the handout data in the submission?

No. You can assume the data will be available under the same path as in the handout folder.

Can you help me solve the task? Can you give me a hint?

As the tasks are a graded part of the class, **we cannot help you solve them**. However, feel free to ask general questions about the course material during or after the exercise sessions.

Can you give me a deadline extension?

We do not grant any deadline extensions!

Can I post on Moodle as soon as have a question?

This is highly discouraged. Remember that collaboration with other teams is prohibited. Instead,

- Read the details of the task thoroughly.
- Review the frequently asked questions.
- If there is another team that solved the task, spend more time thinking.
- Discuss it with your team-mates.

When will I receive the project grades?

We will publish the project grades before the exam the latest.