CS791: Programming Assignment 4

Total Points: 50

October 27, 2025

General Instructions

1. Plagiarism will be strictly penalized, including but not limited to reporting to DADAC and zero in assignments. Use of tools such as ChatGPT, Claude, and Copilot is strictly prohibited. Additionally, if you use external sources (e.g., tutorials, papers, or open-source code), you must cite them properly in your report and comments in the code.
2. Submit a report explaining your approach, implementation details, results, and findings. Clearly mention the contributions of each team member in the report. Submit your code and report as a compressed `<TeamName>_<student1rollno>_<student2rollno>_<student3rollno>.zip` file. Fill a student roll number as `NOPE` if less than 3 members.
3. Start well ahead of the deadline. Submissions up to two days late will be accepted with some penalty, and no marks will be awarded beyond that.
4. Do not modify the environment provided. Any runtime errors during evaluations will result in zero marks. `README.md` provides instructions and tips to set up the environment and run the code.
5. Throughout the assignment, you have to just fill in your code in already existing files. Apart from the report, do not submit any additional models or files. The internal directory structure of your final submission should look as follows:

```
cs791_assgmt4/
  |
  +- acquisition_functions.py [to be changed]
  +- kernels.py [to be changed]
  +- main.py [to be changed]
  +- models.py [to be changed]
  +- train_test.py [to be changed]
  +- utils.py [to be changed]
  +- README.md
  +- PA4_Problem_Statement.pdf
  +- report.pdf [NEW]
```

6. STRICTLY FOLLOW THE SUBMISSION GUIDELINES. Any deviation from these guidelines will result in penalties.

1 Hyperparameter Tuning using Gaussian Process Surrogates

Hyperparameters play a crucial role in model training and can often be the deciding factor between two rival architectures (for example, *Llama-4* vs *Deepseek*). Even a tiny change in a single hyperparameter, like the learning rate, can sometimes lead to substantial performance gains at no additional cost. However, as modern deep learning models come equipped with an overwhelming number of hyperparameters, performing an exhaustive grid search over them is both computationally infeasible and often unnecessary. In this assignment, we will explore whether Gaussian Processes (GPs) can be the promise land that helps address this problem.

Specifically, the goal of this assignment is as follows: You are given a neural network architecture $\mathcal{N}$, a dataset $\mathcal{D} = \{\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}}\}$ and a set of tunable hyperparameters $\mathbf{H}$ which can take values from a predefined search space $\mathcal{H}$. Additionally, you are allowed a total budget of N model training runs, meaning you can train and evaluate the model up to N number of times (**Note that N includes the initial runs done on the randomly sampled hyperparameters during the warm-up phase of Bayesian Optimization**). Let $\mathbf{V}(\mathcal{N}, \mathcal{D}, h)$ denote the evaluation function that first trains $\mathcal{N}$ on $\mathcal{D}_{\text{train}}$ using a specific assignment of hyperparameters $h \in \mathcal{H}$ and then measures how well $\mathcal{N}$ performs on the validation set $\mathcal{D}_{\text{valid}}$. Your mission, should you choose to accept it, is to design a hyperparameter optimization algorithm that, within the training budget N and guided by $\mathbf{V}(\mathcal{N}, \mathcal{D}, h)$, identifies the assignment to $\mathbf{H}$ that yields the best generalization performance of $\mathcal{N}$ on a held-out test set $\mathcal{D}_{\text{test}}$. Note that the algorithm does not have access to $\mathcal{D}_{\text{test}}$.

Interestingly, the objective of this assignment aligns perfectly with a Bayesian Optimization (BO) setup, where $\mathbf{V}$ is the expensive black-box function you are asked to optimize, $h \in \mathcal{H}$ are the values you are allowed to tweak, and GP is the surrogate function (M) you plan to use to model $P(\mathbf{V} \mid \mathcal{D})$. Mathematically, our goal can now be defined as:

$$\boxed{h^* = \arg\max_{h \in \mathcal{H}} \mathbf{V}(\mathcal{N}, \mathcal{D}, h)}$$

Please refer to the class notes for a detailed understanding of GPs, BO, and their constituent components. Here, we will focus primarily on listing out the key elements you are expected to implement. Note that you are not allowed to use any machine learning libraries like `scikit-learn` to offload BO or GP-related operations. You must implement these algorithms on your own from scratch.

1.1 Evaluation Protocol and Metrics

All experiments in this assignment will be conducted primarily on the MNIST dataset. However, during grading, your algorithm might be evaluated on additional datasets. Therefore, ensure that your solutions generalize across datasets, rather than overfitting to MNIST.

We will use the mean validation accuracy (on a $0 - 100$ scale) computed over the examples in $\mathcal{D}_{\text{valid}}$ as the output from the evaluation function $\mathbf{V}$, which BO uses to sample the next point. The same metric will also be used at the end of the optimization process to assess how well the final hyperparameter configuration performs on the held-out test set $\mathcal{D}_{\text{test}}$.

1.2 Task 0: Building the Training and Validation Pipeline [10 pts]

In this task, you will implement the function $\mathbf{V}$, which we will use as the blackbox function during Task 1. Specifically, complete the function `train_and_test_NN` in `train_test.py`, which should perform one full run of training followed by validation for the model $\mathcal{N}$. At the end, the function should return the mean validation accuracy of the trained model. The exact sequence of steps is as follows:

- STEP 1: Create dataloaders for both training and validation datasets, with proper batching.
- STEP 2: Initialize the following components:
  1. the model (`SimpleNN` from `models.py`) with the specified hidden layer size and dropout rate.
  2. the optimizer with the given learning rate and weight decay
  3. the cross-entropy loss function
- STEP 3: Train the model for the specified number of epochs. In each epoch, iterate over the entire training dataset, computing the loss and performing backpropagation for each batch.

- STEP 4: Evaluate the trained model on the validation dataset and return the mean validation accuracy.

1.3 Task 1: Exploring Kernel and Acquisition functions [25 pts]

We will be optimizing on the following hyperparameters with the given ranges:-

- Hidden Layer Size - $\{100, 200, 300, 400, 500\}$
- Training Epochs - $[1, 10]$
- Learning Rate - $[10^{-5}, 10^{-1}]$
- Batch Size - $\{16, 32, 64, 128, 256\}$
- Dropout Rate - $[0, 0.5]$
- Weight Decay - $[10^{-6}, 10^{-2}]$

Hint: As some of the hyperparameters have a continuous space, for choosing the next sample in the Baysian Optimization process you can assume the space to be some number of discrete equally spaced points in the linear or logarithmic domain.

1.3.1 Kernels

Implement the following kernel types manually in `kernels.py`:

- **Radial Basis Function (RBF) Kernel:** $k(x, x') = \sigma_f^2 \exp\left(-\frac{\|x-x'\|^2}{2\ell^2}\right)$, where ℓ is the length scale and σ_f is the signal variance.
- **Matérn Kernel ($\nu = 1.5$):**

$$k(x, x') = \sigma_f^2 \left(1 + \frac{\sqrt{3}\|x - x'\|}{\ell}\right) \exp\left(-\frac{\sqrt{3}\|x - x'\|}{\ell}\right),$$

  where ℓ is the length scale and σ_f is the signal variance.

- **Rational Quadratic Kernel:**

$$k(x, x') = \sigma_f^2 \left(1 + \frac{\|x - x'\|^2}{2\alpha\ell^2}\right)^{-\alpha},$$

  where ℓ is the length scale, σ_f is the signal variance, and α is the scale mixture parameter (set $\alpha = 1$).

Optimize the kernel hyperparameters (ℓ and σ_f) by maximizing the log-marginal likelihood using a grid search approach (optional for better predictions) by implementing the functions in `utils.py`.

1.3.2 Acquisition Functions

Implement the following Acquisition Functions in `acquisition_functions.py`:

- **Expected Improvement (EI):** $\mathrm{EI}(x) = (\mu(x) - f(x_{\text{best}}) - \xi)\Phi(z) + \sigma(x)\phi(z)$, where $z = \frac{\mu(x)-f(x_{\text{best}})-\xi}{\sigma(x)}$, Φ is the cumulative distribution function, and ϕ is the probability density function of the standard normal distribution, with $\xi = 0.01$.
- **Probability of Improvement (PI):** $\mathrm{PI}(x) = \Phi\left(\frac{\mu(x)-f(x_{\text{best}})-\xi}{\sigma(x)}\right)$, with $\xi = 0.01$.

1.3.3 Bayesian Optimization

In `main.py`, implement the Bayesian Optimization loop by using the `gaussian_process_predict` function from `utils.py`. To do so, you will first run $N_{\text{warmup}} = 10$ warm-up steps, during which the hyperparameter values are randomly sampled from their respective ranges. After the warm-up phase, proceed with the BO using GPs and perform $N - N_{\text{warmup}}$ iterations. Your final evaluation on the held-out test set would be on the hyperparameter configuration that achieved the best mean validation accuracy among all N samples.

For each combination of the kernel and acquisition function (i.e., six configurations), report the final hyperparameter values along with the corresponding accuracy on the test dataset. Additionally, for each configuration, you must also plot the progression of the validation accuracy over the BO iterations. For these experiments, you can use $N = 25$. Finally, using the best kernel and acquisition function, analyze the impact of the budget N by experimenting with $N = \{15, 25, 50\}$ and report your findings. Provide analysis and explanations for all the observations.

1.4 Task 2: Exploring BO for a different architecture [15 pts]

Based on your exploration in the previous task, you now have a good understanding of GPs, BO, and their constituent components. In this task, you will apply that knowledge to optimize a convolutional neural network (CNN). You are free to design your own architecture and introduce new hyperparameters of your choice; however, the total number of parameters should not exceed 1 million. You are at liberty to decide the set of hyperparameters you want to optimize, their corresponding ranges, the appropriate definitions for kernel and acquisition functions, and you can even design your own BO algorithm. For this task, use $N_{\text{warmup}} = 10$ and $N = 50$.

Note that you are not allowed to use any additional datasets or pretrained models, and the evaluation setup must remain strictly untouched. Modifying random seeds, datasets, or evaluation metrics, or using test as validation, will result in disqualification (0 points). Additionally, your experiments must respect the budgets set in place. Exceeding these will also result in disqualification.

In your report, clearly describe your modifications, justify your design choices, and present the corresponding evaluation metrics and plots. During grading, submissions will be evaluated and ranked on a leaderboard. We will release this leaderboard during grading. The top 5 teams will be awarded full 15 points, the next 10 teams will receive 10 points, and the remaining teams that successfully attempt this task will be awarded 5 points. Teams that do not attempt this task will receive 0 points.