Bayesian Optimization

Bhatti Roben
Chiloiro Marco
Panagiotakis Konstantinos
Sapkas Michail

Our Goal: Hyperparameter Optimization

- Given a classification problem, the goal is to **fine-tune** the **hyperparameters** for loss minimization of the model

- Challenges: Traditional optimization methods struggle due to the high cost of evaluating

- Benchmark accuracy with a grid-search algorithm on a CNN, time with a random grid search

Bayesian Optimization

$$f : \chi^{(d)} \rightarrow \mathbb{R}$$

- Objective: Optimize the function **f = error of classifier**, over the set X (hyperparameter space)

- We operate under the assumption that the objective function is drawn from a Gaussian Process, our **surrogate function**

Gaussian processes

$$\mathbf{P}(X) = \underbrace{\frac{1}{\sqrt{(2\pi)^k |\Sigma|}}}_{\text{Normalization Term}} \cdot \exp\underbrace{\left(-\frac{1}{2}(X-\mu)^T\Sigma^{-1}(X-\mu)\right)}_{\text{likelihood of observing a particular value of X as it moves away from the mean μ, scaled by the covariance matrix Σ}};$$

Multivariate Gaussian Distribution: This is a probability distribution that describes the likelihood of observing a set of correlated variables. It's characterized by a mean vector (μ) and a covariance matrix (Σ). In the provided context, it refers to a Gaussian distribution in multiple dimensions (d dimensions).

The **covariance matrix** describes the relationship between multiple variables.

Typically, we use the all-zeros vector for the mean μ, and replace the covariance matrix Σ with a Kernel function K.

- Key Insight: Leverage the gaussian process model to **compute** the **conditional distribution**

$$\mathbf{P(f(x) \mid f(x_1),f(x_2),\ldots,f(x_D))}$$

$$f(x_*)|(f(x_1) = y_1, f(x_2) = y_2, \cdots, f(x_D) = y_D) \sim \mathcal{N}(\mathbf{k}_*^T \Sigma^{-1} y, K(x_*, x_*) - \mathbf{k}_*^T \Sigma^{-1} \mathbf{k}_*)$$

Acquisition Functions

Our strategies for **exploring** new points in the search process.

depend on three factors: the mean of the latent variable $f(x^*)$, the standard deviation of $f(x^*)$, and the optimal value observed thus far during the optimization, denoted as y_{best}

We implemented two ways:

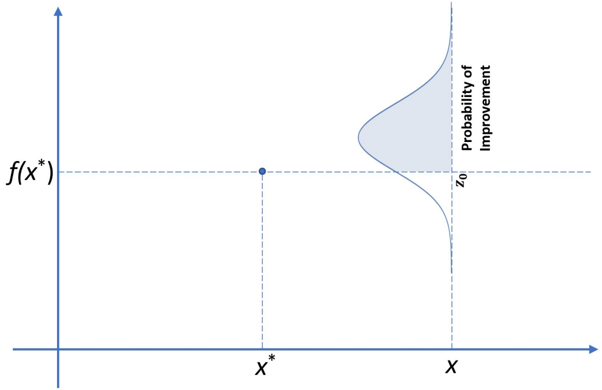
- Probability of Improvement
- Expected Improvement

Probability of Improvement

One intuitive strategy is to maximize the probability of improving over the best current value x*

- We can define **improvement** as: $I(x) = max(f(x) - f(x^*), 0)$
- For each candidate x we assign the probability of $I(x)>0$
- Each candidate has a Gaussian distribution attached
- Under GP this can be computed analytically as:

$$a_{\mathrm{PI}}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta) = \Phi(\gamma(\mathbf{x})), \qquad \gamma(\mathbf{x}) = \frac{f(\mathbf{x}_{\mathrm{best}}) - \mu(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}{\sigma(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}.$$



Expected Improvement

- PI does not consider the magnitude of the improvement
- Instead of looking at the improvement *I(x)* which is a random variable, we will instead calculate the “Expected Improvement” as function of x

$$\underbrace{a_{\mathrm{EI}}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}_{\text{acquisition function depending on the previous observations, as well as the GP hyperparameters}} = \underbrace{\sigma(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}_{\text{predictive variance function}}\,(\underbrace{\gamma(\mathbf{x})\,\Phi(\gamma(\mathbf{x}))}_{\gamma(\mathbf{x}) = \frac{f(\mathbf{x}_{\mathrm{best}}) - \mu(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}{\sigma(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)}} + \underbrace{\mathcal{N}(\gamma(\mathbf{x})\,;\,0, 1)}_{\text{normal (Gaussian) distribution with mean 0 and standard deviation 1}})$$

Why Must the Function be a Kernel?

Kernels in Gaussian processes possess two crucial properties:

- **symmetry**
- **positive semidefiniteness**

Symmetry Requirement Necessity: Symmetry is essential because $f(x_i) \cdot f(x_j)$ is a symmetric expression. Ensures that the ordering of x_i and x_j does not impact the result.

Positive Semidefiniteness Requirement Necessity: Positive semidefiniteness is vital due to the nature of covariance matrices. Any valid covariance matrix must be positive semidefinite, including infinite-dimensional cases.

Calculate the new mean and standard deviation

The posterior mean and variance evaluated at any point x represent the model's prediction and uncertainty in the objective function at the point x. These posterior functions are used to select the next query point x_{n+1}

$$\mu_n(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - \mathbf{m})$$

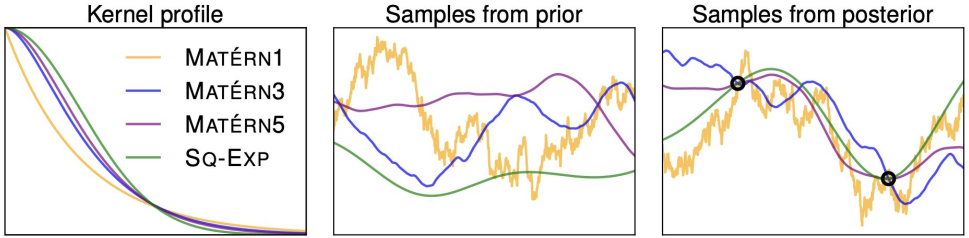
$$\sigma_n^2(\mathbf{x}) = k(\mathbf{x}, \mathbf{x}) - \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}(\mathbf{x})$$

The Kernel Matrix

$$\Sigma = \begin{bmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots & K(x_1, x_D) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_D, x_1) & K(x_D, x_2) & \cdots & K(x_D, x_D) \end{bmatrix}$$

Kernel Function (or Covariance Function): The kernel function, denoted as $K(\cdot, \cdot)$, defines the relationship between pairs of input points in the space X. For any two points $x_i, x_j \in X$, the kernel function $K(x_i, x_j)$ computes the covariance (or similarity) between the corresponding outputs $f(x_i)$ and $f(x_j)$ of the Gaussian process.

ARD Matern 5/2 kernel



$$K_{\mathrm{M52}}(\mathbf{x},\mathbf{x}') = \underbrace{\theta_0}_{\text{s (Signal Variance)}} \left(1 + \underbrace{\sqrt{5r^2(\mathbf{x},\mathbf{x}')}}_{\text{distance between } x_1 \text{ and } x_2 \text{ scaled by the length scales l}} + \frac{5}{3}r^2(\mathbf{x},\mathbf{x}')\right) \exp\left\{-\sqrt{5r^2(\mathbf{x},\mathbf{x}')}\right\}$$

$$r^2(\mathbf{x},\mathbf{x}') = \sum_{d=1}^{D} (x_d - x'_d)^2/\theta_d^2.$$

Particularly useful when modeling processes with **smooth** but non-differentiable functions.

s: Scalar parameter representing the variance or amplitude of the signal. Controls the overall amplitude of the kernel function.

l: Scales the distance between input points. A larger length scale results in smoother functions.

Sample these

The Algorithm

Algorithm 1 Bayesian optimization with Gaussian process prior

input: loss function f, kernel K, acquisition function a, loop counts N_{warmup} and N

▷ warmup phase

$y_{\text{best}} \leftarrow \infty$

for $i = 1$ **to** N_{warmup} **do**

  select x_i via some method (usually random sampling)

  compute exact loss function $y_i \leftarrow f(x_i)$

  if $y_i \leq y_{\text{best}}$ **then**

    $x_{\text{best}} \leftarrow x_i$

    $y_{\text{best}} \leftarrow y_i$

  end if

end for

for $i = N_{\text{warmup}} + 1$ **to** N **do**

  update kernel matrix $\Sigma \in \mathbb{R}^{i \times i}$ according to (1)

  let $\mu(x_*)$ and $\sigma(x_*)$ denote the expected value and standard deviation, respectively, of $f(x_*)$ under the Gaussian process model, conditioned on all the previous observations of $f(x_i) = y_i$

  $x_i \leftarrow \arg\min_{x_*} \ a(\mu(x_*), \sigma(x_*), y_{\text{best}})$

  compute exact loss function $y_i \leftarrow f(x_i)$

  if $y_i \leq y_{\text{best}}$ **then**

    $x_{\text{best}} \leftarrow x_i$

    $y_{\text{best}} \leftarrow y_i$

  end if

end for

return x_{best}

warmup

sampling

The Algorithm - input

Algorithm 1 Bayesian optimization with Gaussian process prior

input: loss function f, kernel K, acquisition function a, loop counts N_{warmup} and N

▷ warmup phase

$y_{\text{best}} \leftarrow \infty$

- **Goal** - find the hyperparameter combination that minimizes the loss
- **Input:**
  - **Training Loss** as the function f to be minimized
  - **Kernel 5/2** with l,s as hyper-hyper params
  - Aquisition function **EI** for hyperparameter selection
  - **Number of Warmup** loops
  - **Number of Sampling** loops

The Algorithm - warmup

for $i = 1$ **to** N_{warmup} **do**
  select x_i via some method (usually random sampling)
  compute exact loss function $y_i \leftarrow f(x_i)$
  if $y_i \leq y_{\text{best}}$ **then**
    $x_{\text{best}} \leftarrow x_i$
    $y_{\text{best}} \leftarrow y_i$
  end if
end for

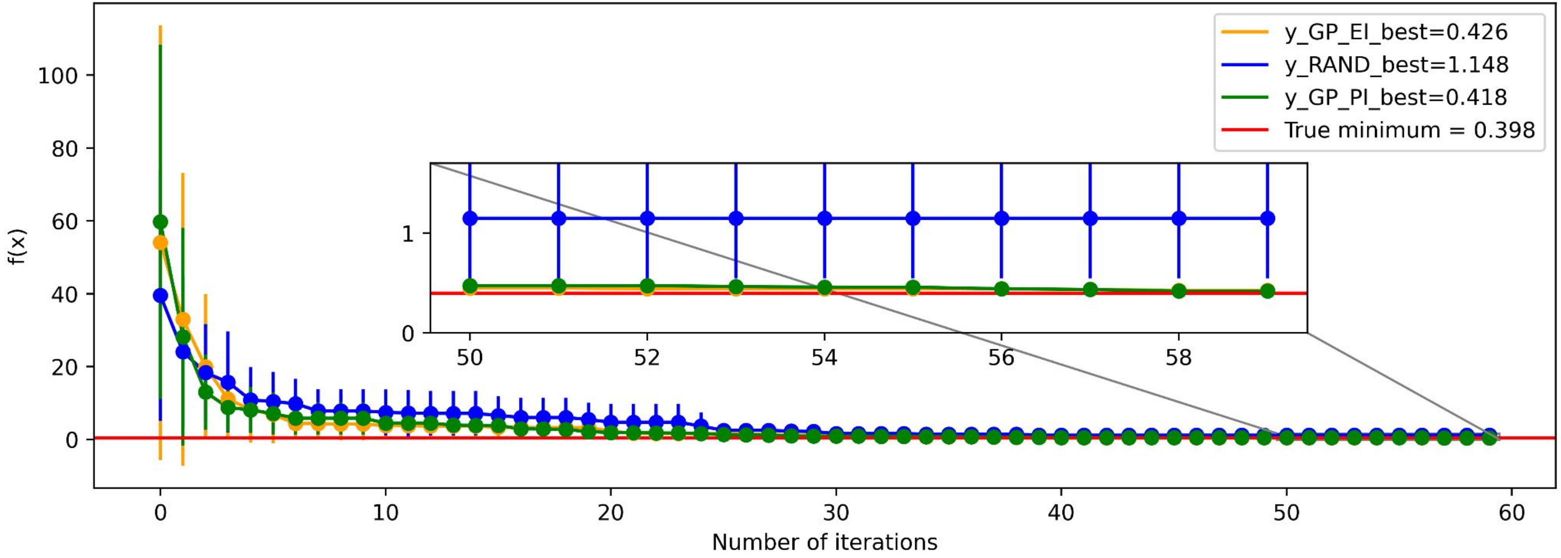
- Loop until **last** number of **warmup loops**
- Use **random shuffle** to select hyperparameters
- Save best hyperparameter combination that minimize loss

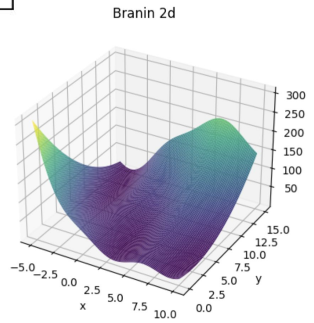
The Algorithm - sampling

- Loop from last number of warmup loops until end of **sampling loops**
- Update the **kernel matrix**
- Calculate the mean and std of the loss function at the new hyperparameter point
- Use **acquisition function EI** to select next hyperparameter point
    - Select hyperparameter combination that maximizes acq func.
- Save best hyperparameter combination that minimizes loss

for $i = N_{\text{warmup}} + 1$ **to** N **do**
  update kernel matrix $\Sigma \in \mathbb{R}^{i \times i}$ according to (1)
  let $\mu(x_*)$ and $\sigma(x_*)$ denote the expected value and standard deviation, respectively, of $f(x_*)$ under the Gaussian process model, conditioned on all the previous observations of $f(x_i) = y_i$
  $x_i \leftarrow \arg\min_{x_*} \; a(\mu(x_*), \sigma(x_*), y_{\text{best}})$
  compute exact loss function $y_i \leftarrow f(x_i)$
  if $y_i \leq y_{\text{best}}$ **then**
    $x_{\text{best}} \leftarrow x_i$
    $y_{\text{best}} \leftarrow y_i$
  end if
end for
return x_{best}

2D hyperparameter test on Branin function



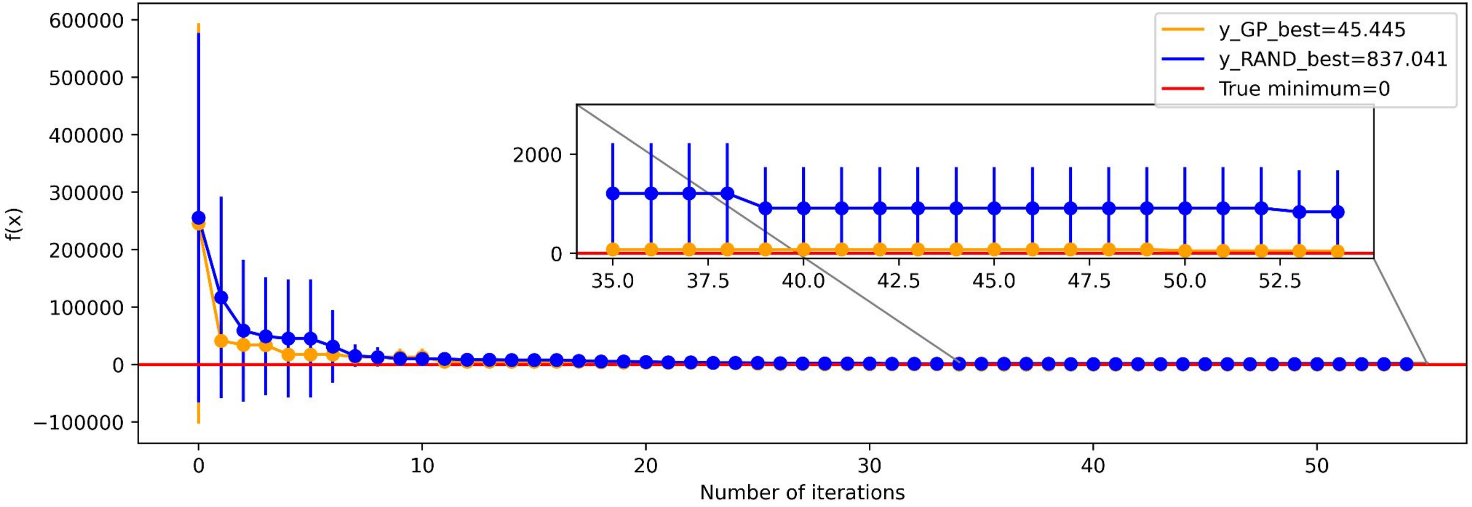


$$f(x_1, x_2) = a(x_2 - bx_1^2 + cx_1 - r)^2 + s(1-t)\cos(x_1) + s.$$

The typical parameter values are $a = 1, b = 5.1/(4\pi^2), c = 5/\pi, r = 6, s = 10$ and $t = 1/(8\pi)$. The function is usually evaluated over the square $x_1 \in [-5, 10]$, $x_2 \in [0, 15]$.

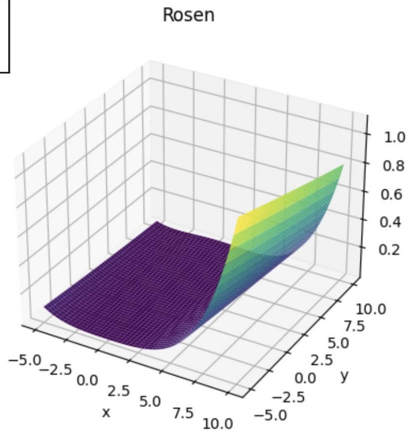
Branin's global minimum lies at [-pi,12], where it evaluates to 0.398

3D hyperparameter test on Rosenbrock function



$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} \left[100 \left(x_{i+1} - x_i^2 \right)^2 + \left(1 - x_i \right)^2 \right]$$

Rosenbrock's minimum is at x=1 for each dimension, yielding a function value of zero adaptable to any dimension

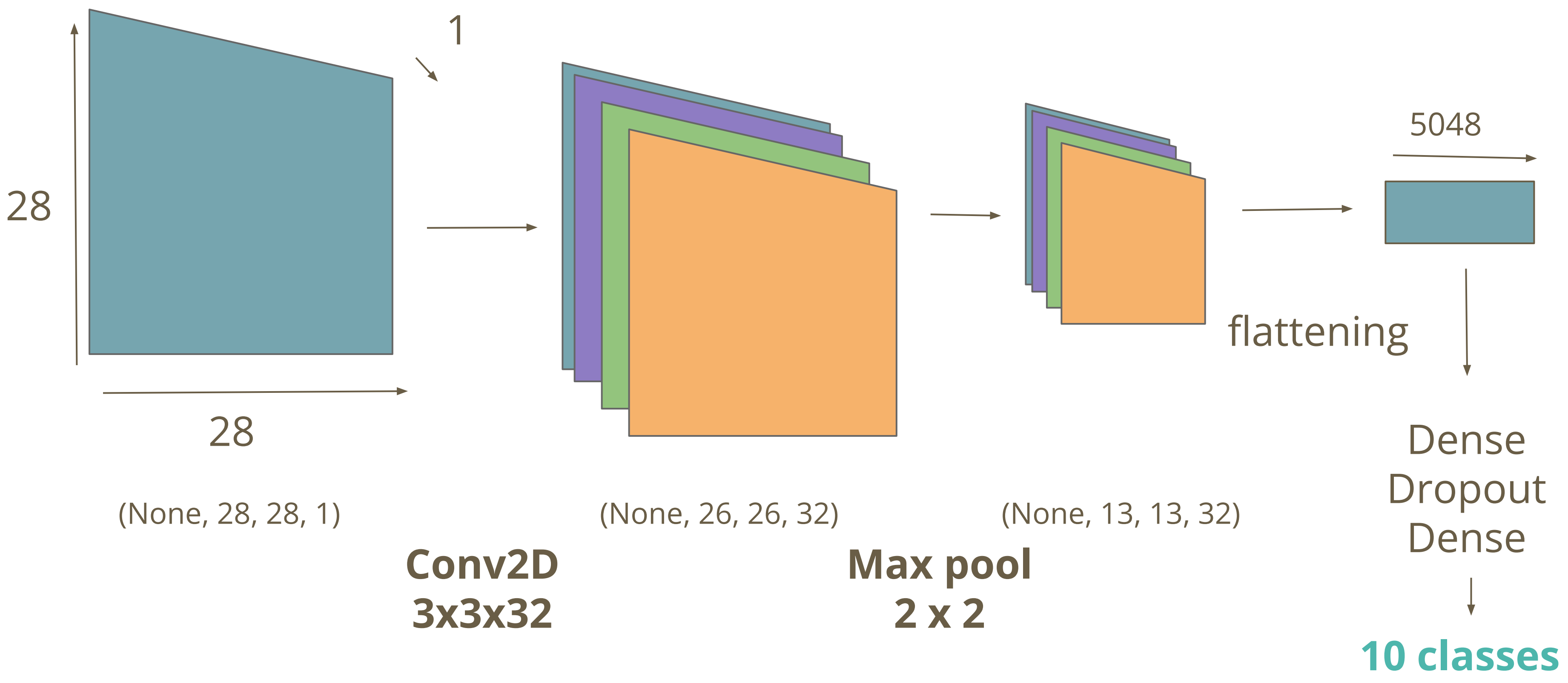


Applied Bayesian Optimization to CNN

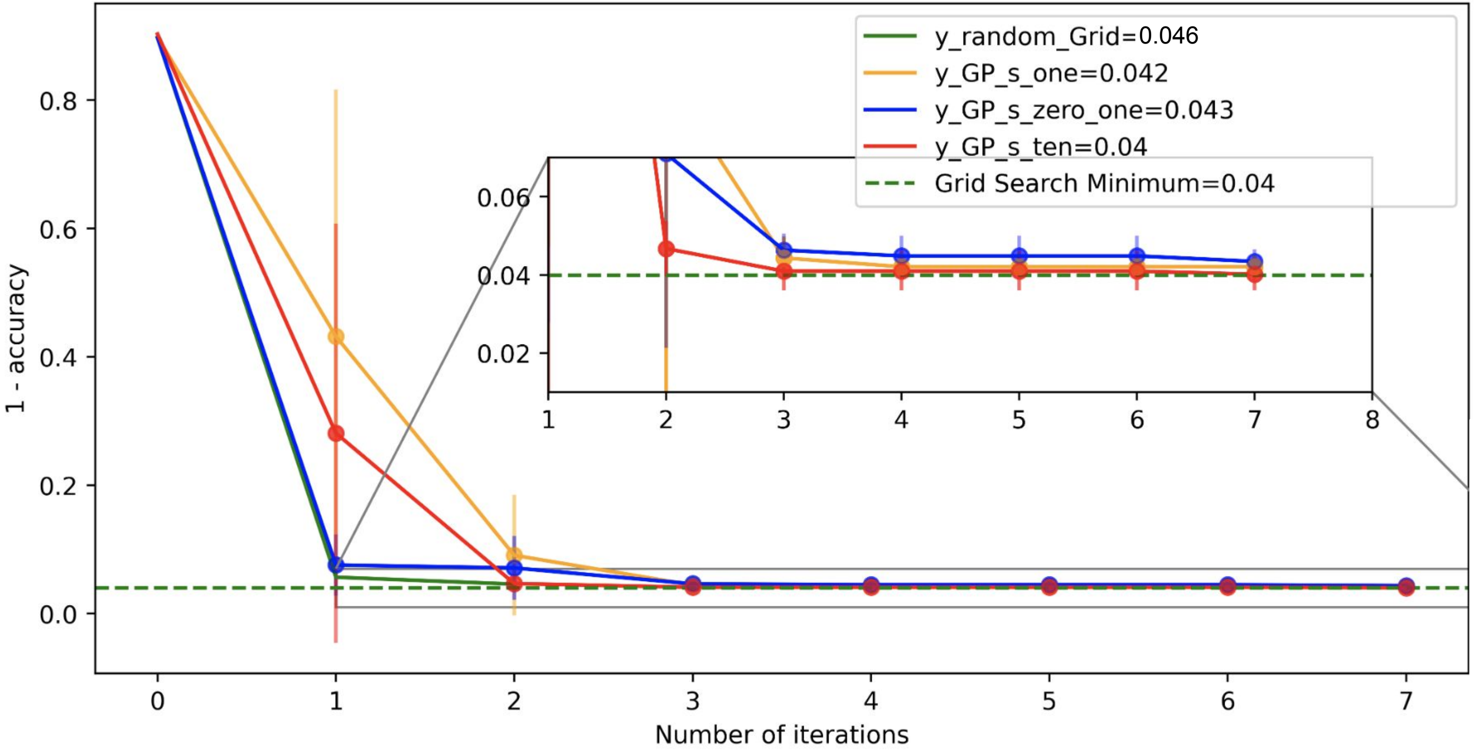
- **Hyperparameter space**:
    - { "dropout_rate": [0.8, 0.6, 0.4, 0.2, 0.1], "learning_rate": [0.5, 0.1, 0.01, 0.001]}
- n_test = 5
- n_sample = 6
- n_wu = 2

- **Recall** - Optimize the function f: X→R over the set X (hyperparameter space)
- f -> loss function
- X -> set of hyperparameter combinations

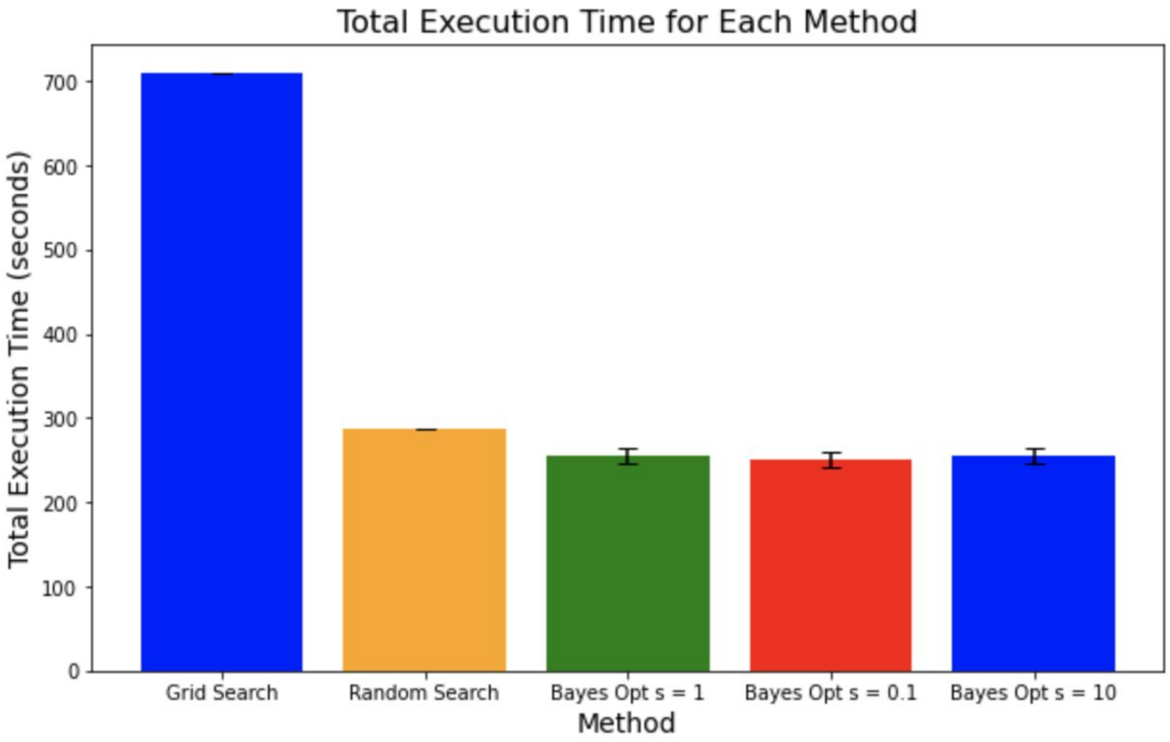
Simple CNN on MNIST



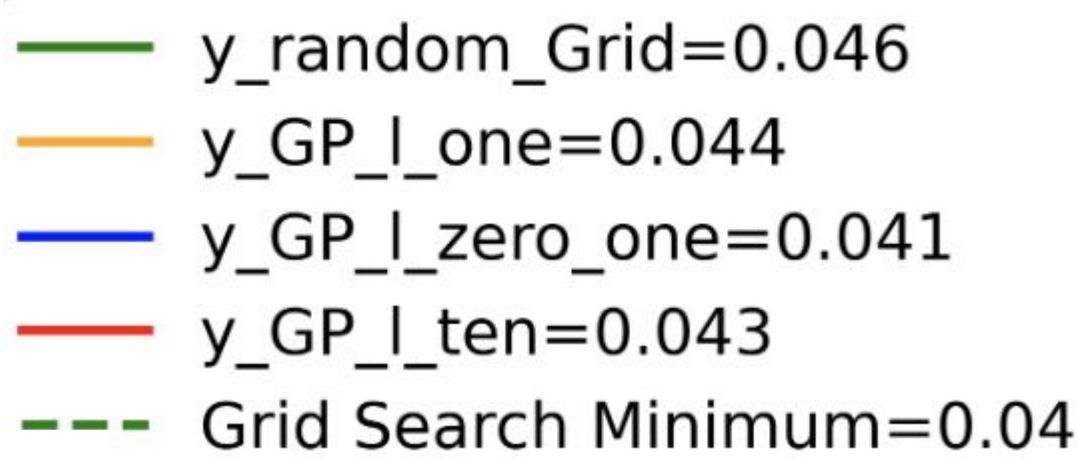
Small hyper-space - Changing s



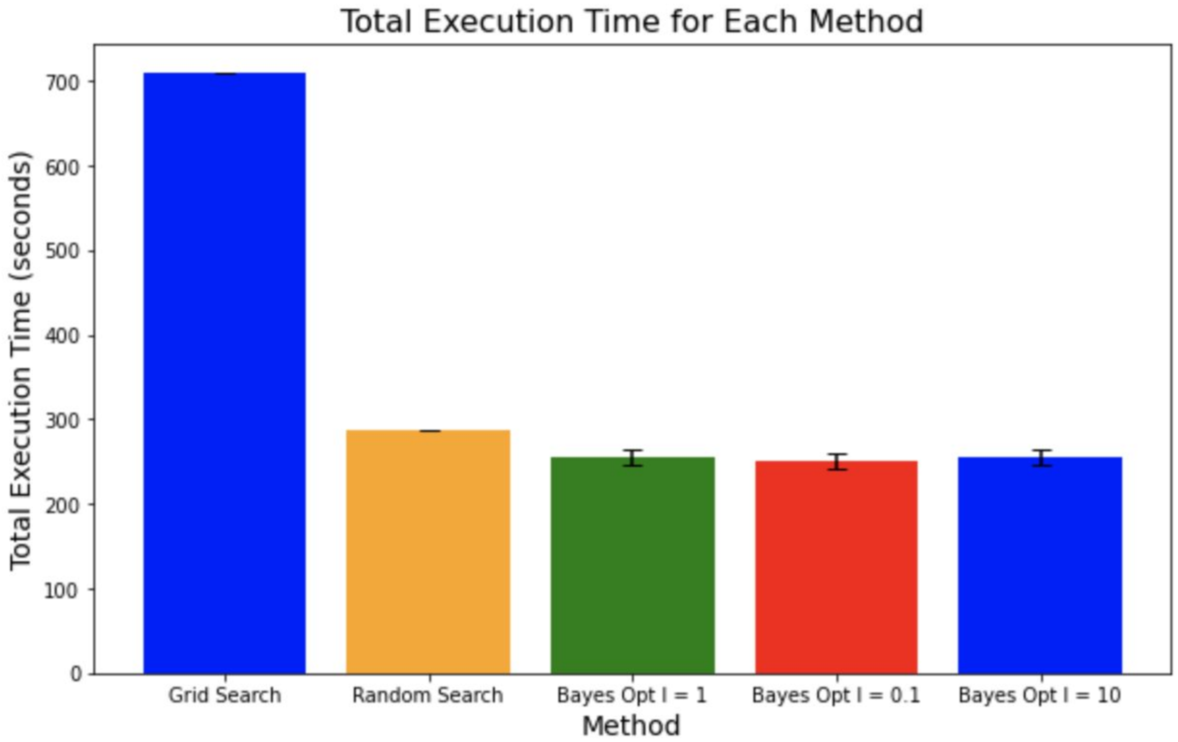
Small hyper-space - Changing s



Small hyper-space - Changing l



Small hyper-space - Changing l



Kernel hyperparameters

Bayesian Optimization using Gaussian Process has hyperparameters itself (let's say hyper-hyperparameter), i.e. the Kernel ones.

There are two main possible ways to treat them:

- **Point estimation** by minimizing the marginal likelihood
- Computing the **integrated acquisition function** by using MCMC to sample the posterior over hyper-hyperparameters and have a Monte Carlo estimate.

Integrated acquisition function

Integrated acquisition function:

$$\hat{a}(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}) = \int a(\mathbf{x}\,;\,\{\mathbf{x}_n, y_n\}, \theta)\, p(\theta \,|\, \{\mathbf{x}_n, y_n\}_{n=1}^{N})\, \mathrm{d}\theta$$

MC estimate using M samples from the posterior:

$$\mathbb{E}_{\theta|\mathcal{D}_n}\left[\alpha(x;\theta)\right] \approx \frac{1}{M}\sum_{i=1}^{M}\alpha(x;\theta_n^{(i)})$$

Posterior

- Posterior: $$p(\theta \,|\, \mathcal{D}_n) = \frac{p(\mathbf{y} \,|\, \mathbf{X}, \theta) p(\theta)}{p(\mathcal{D}_n)}$$
- Likelihood: $$p(\mathbf{y}|\mathbf{X}, \theta) = \mathcal{N}(\mathbf{y}; \mathbf{0}, \Sigma_\theta)$$
- Uniform prior $$p(\theta) = \mathcal{U}(L)$$

Metropolis algorithm problems

To sample from the posterior, one could use the Metropolis algorithm.

Problems:

- Proposal distribution (jump size)
- Burn-in phase (convergence)
- Thinning (autocorrelation)

Vihola algorithm

Instead of employing the Metropolis algorithm, we opted for the Vihola algorithm.

This approach enabled us to focus solely on two parameters, irrespective of the dimensionality of the sample under consideration. Moreover, it uses an iterative proposal distribution.

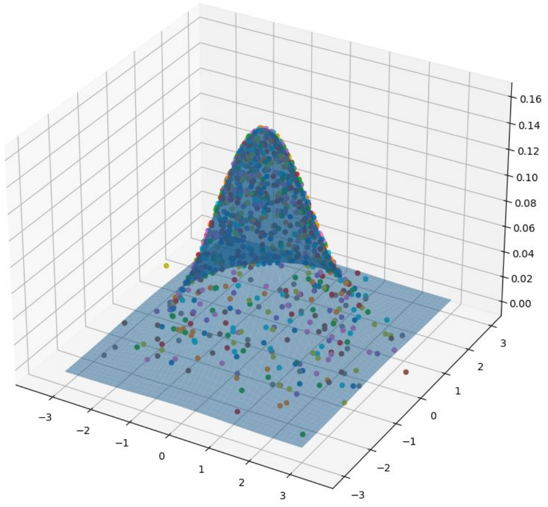
- Learning rate $\eta = 0.5$
- Desired mean acceptance rate $\alpha = 0.7$

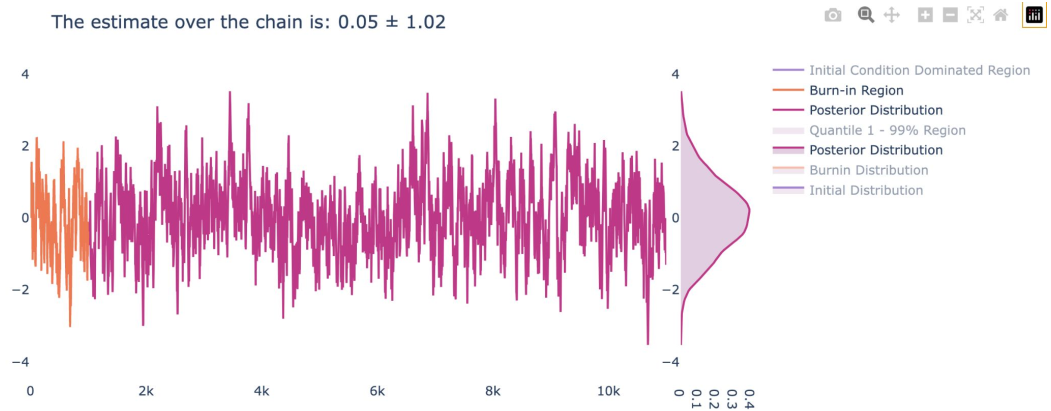
Vihola algorithm

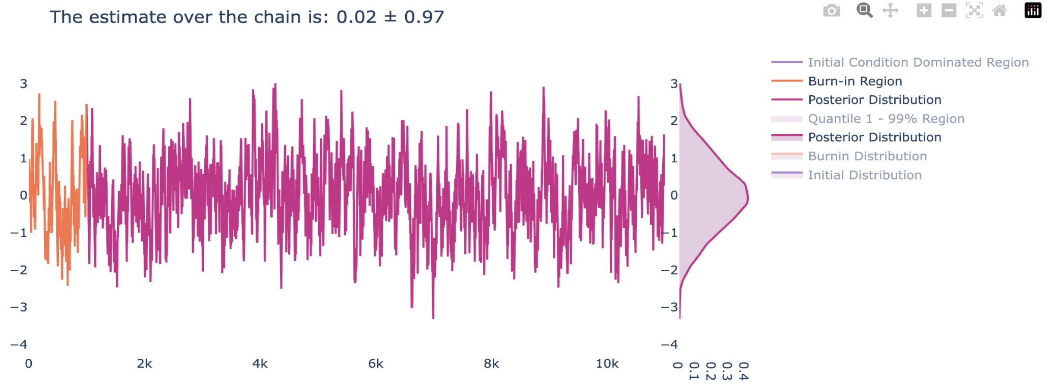
Steps:

1. Cholesky decomposition: $$\Sigma_{n-1} = L_{n-1} L_{n-1}^T$$
2. Jump proposal: $$\theta_n^* = \theta_{n-1} + L_{n-1}\mathbf{u}_n \qquad \mathbf{u}_n \sim \mathcal{N}(0,1)$$
3. Metropolis acceptance probability α_n
4. Update covariance matrix $$\Sigma_n = L_{n-1}\left(\mathbb{1} + \eta(\alpha_n - \alpha)\frac{\mathbf{u}_n \mathbf{u}_n^T}{||\mathbf{u}_n||^2}\right) L_{n-1}^T$$

Metropolis on toy example

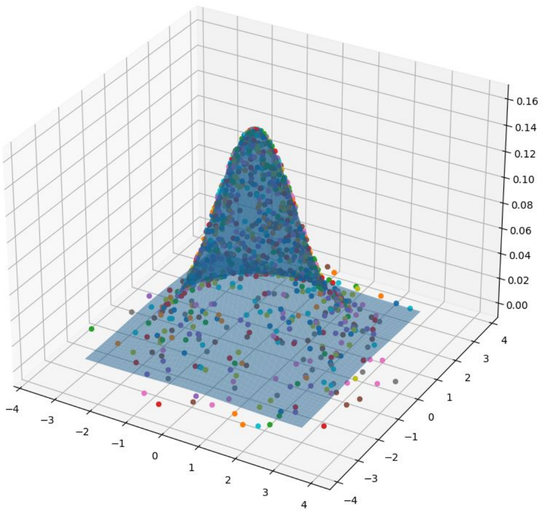


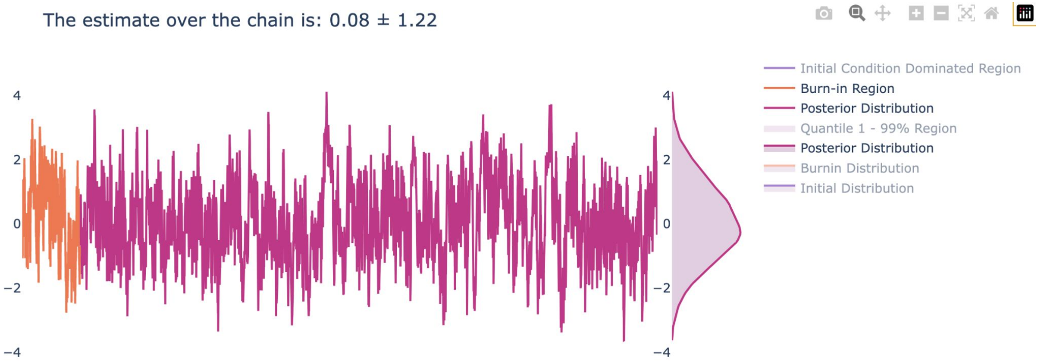


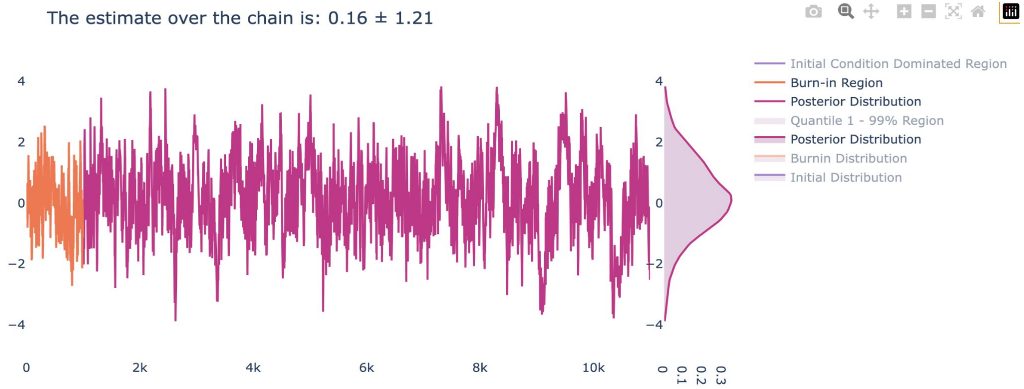


Mean acceptance rate = 0.64

Vihola on toy example







Mean acceptance rate = 0.72

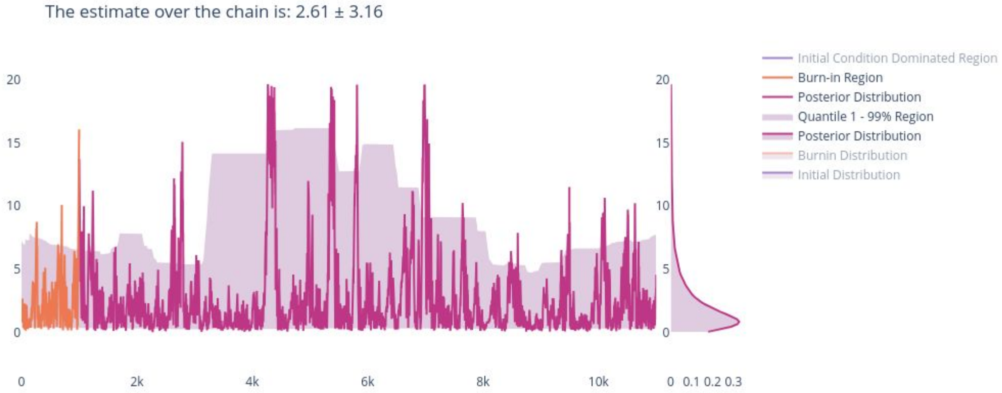
Sampling simulation (case 1)

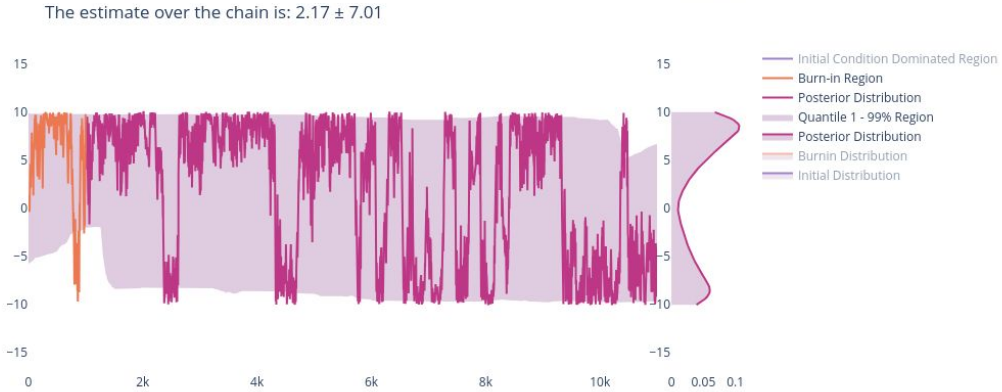
Two sampled points:
x={0.1, 0.01}; y={0.8, 0.6}

s [0, 20]

l [-10, 10]

Sampling simulation (case 2)



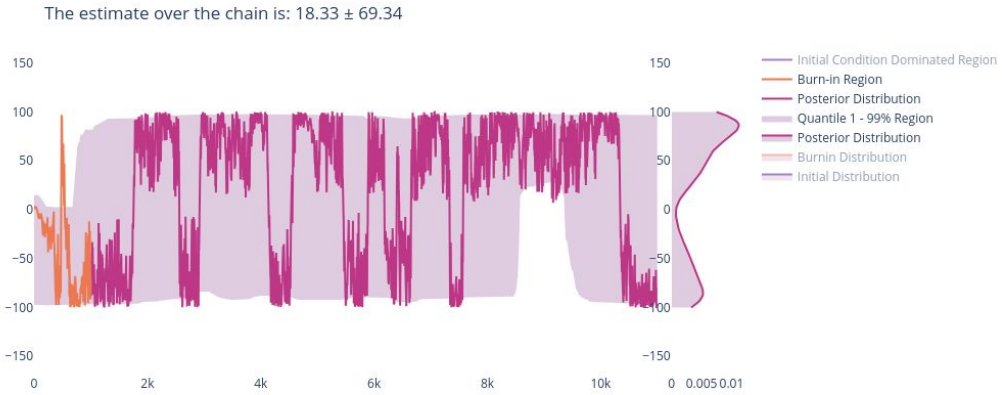


Two sampled points:
x={3, 4}; y={0.01, 0.01}

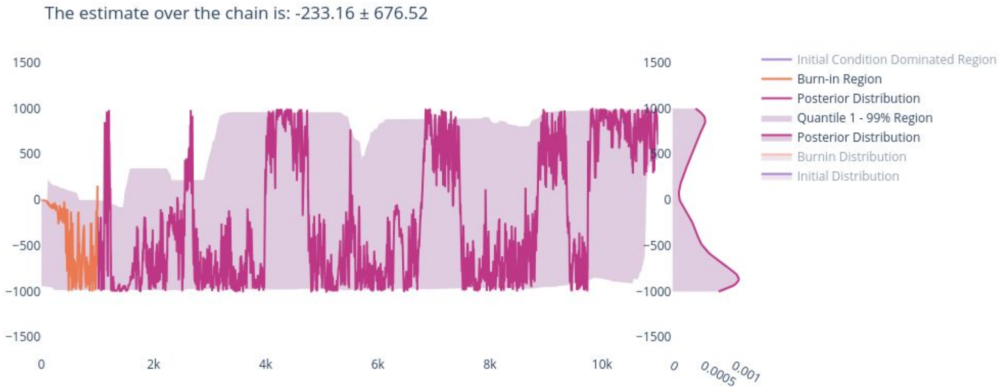
s [0, 20]

l [-10, 10]

Sampling simulation (case 2)



- [-100, 100]



- [-1000, 1000]

Sampling problems

- Sampling strongly depends on the hyper-hyperparameters bounds (prior)
- Likelihood is a not-trivial distribution, particularly for large dimensionality
- More hyper-hyperparameters than before (bounds, learning rate, mean acceptance rate, burn-in, thinning)

Minimizing the negative log likelihood

- Invoke a numerical optimizer to find the parameters that minimize the likelihood of the multivariate Gaussian

$$\log p(\mathbf{y}|X, \boldsymbol{\theta}) = -\frac{1}{2}\mathbf{y}^\top K_y^{-1}\mathbf{y} - \frac{1}{2}\log|K_y| - \frac{n}{2}\log 2\pi,$$

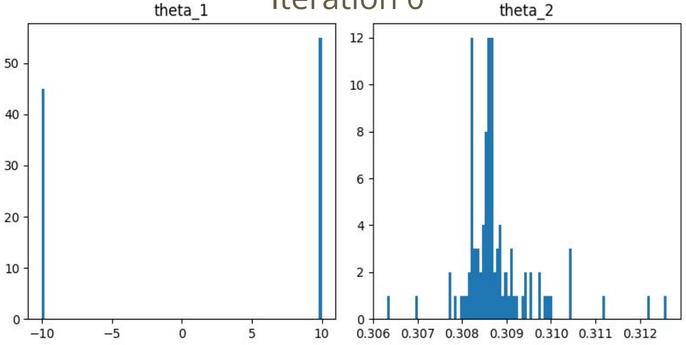
Optimization Process

- After each **f(x)** evaluation
- Initialize N walkers randomly on the landscape
- Acquire the values
- Calculate the histogram
- Pick the value with the maximum frequency

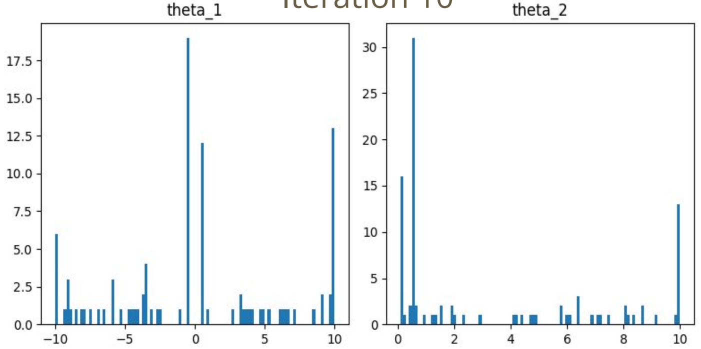
- Results are not encouraging. Walkers don’t converge, and bounding the optimization problem becomes necessary.
- Many possible approaches and optimization methods

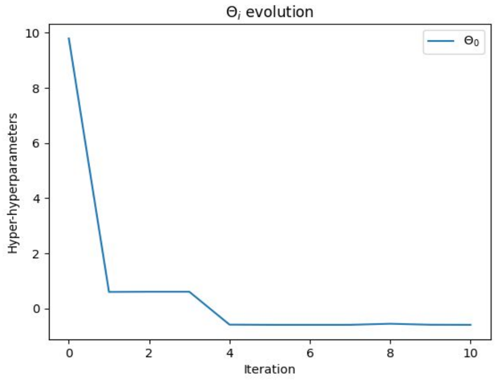
Tuning the hyper-hyperparameters

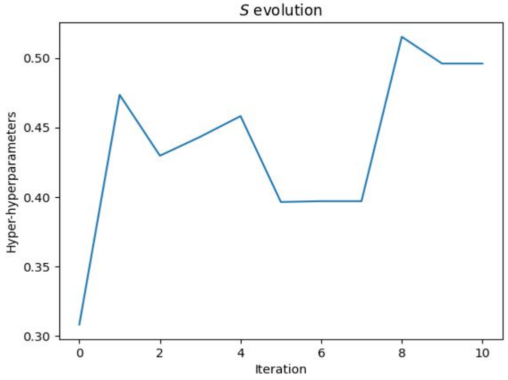
Iteration 0



Iteration 10







Conclusions

- Applied Bayesian Optimization on simple CNN task with l,s fixed by hand
- Applied MCMC and Point Estimation on the sampling of hyper-hyperparameters
- MCMC and Point Estimate fail to converge values for l,s
- Future work:
    - Implement MCMC/Point estimate for hyper-hyperparameter sampling