Bayesian Optimization for Hyperparameters Tuning in Neural Networks

Presentazione finale di Progetto Interno
equipollente a Tirocinio

Docente Valutatore:
Prof. Simone Scardapane

Agenda

1 Introduction

2 Neural Networks Basics

3 Bayesian Optimization

4 Test Function Outcomes

5 Case Study & Considerations

6 Conclusions

Introduction

What is Mathematical Programming?

Is a subfield of Applied Mathematics that focuses on finding the optimal value within an allowed set for a given function of interest.

It spans in several field such as:

- Linear and Nonlinear Programming
- Multi-Objective and Stochastic Programming
- Integer Programming

Where does Bayesian Optimization falls?

Bayesian Optimization is a **derivative-free** optimization technique designed to identify the optimal solution for **expensive-to-evaluate black-box** functions, while also effectively handling **stochastic noise** in the evaluations.

1. The **input should not be excessively** large, typically a **dimensionality at most 20** is suitable for practical applications.
2. The feasible set must be **easy to assess membership** such as a **hyper-rectangle** or a simplex.
3. Can work with **integer variables** effortlessly.
4. Handles **multi-objective** optimization.

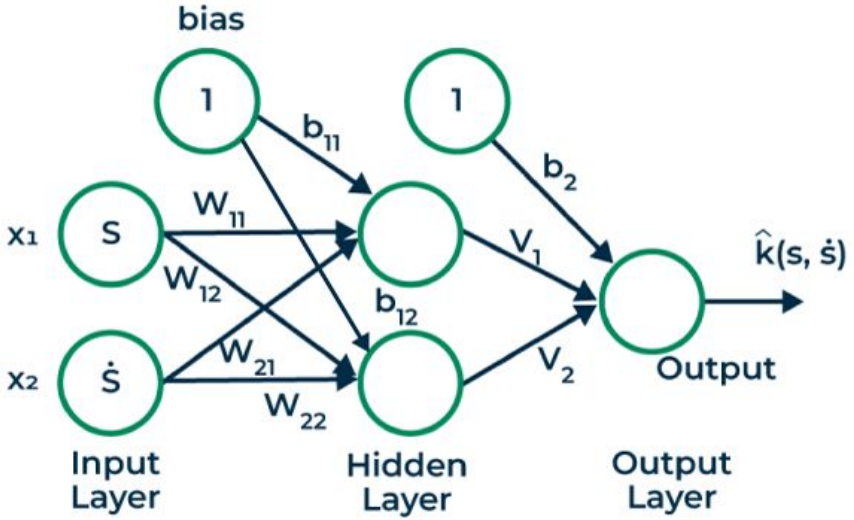
Our Goal

We aim to increase the **performance** of a **Neural Network** purposed for **image classification** tasks on the CIFAR10 dataset.

Neural Networks Basics

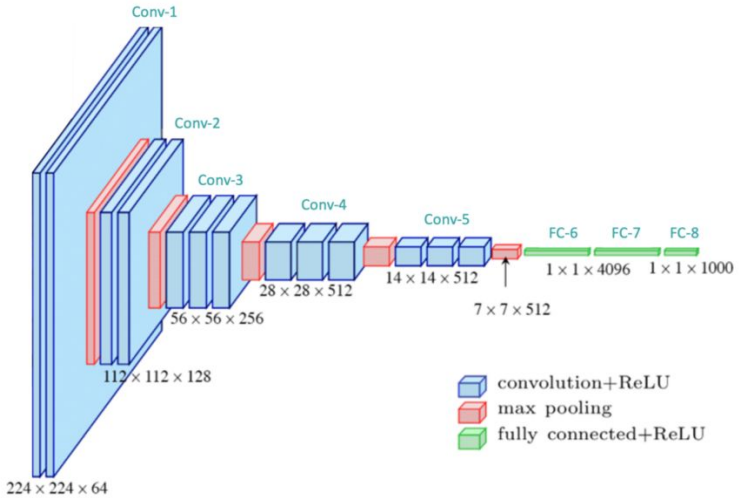
How is a Neural Network composed?

1. Layers (Fully Connected or Convolutional)
2. Weights and Biases
3. Activation Functions (Non-Linearity)



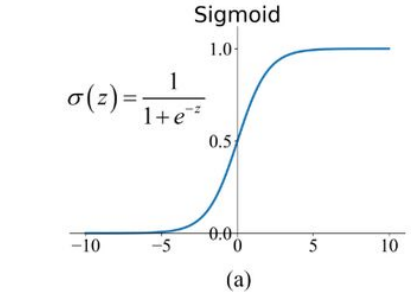
A Convolutional Neural Network (CNN)

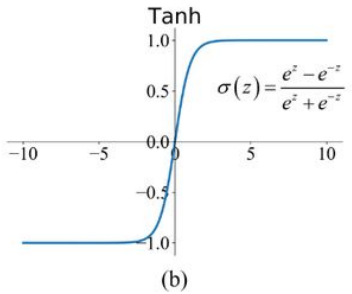
CNNs are suited for image related tasks, where **convolutional** layers are used to **extract spatial features**, **max pooling** is employed for **dimensionality reduction** and then the data is **processed** through a series of **fully connected layers**.

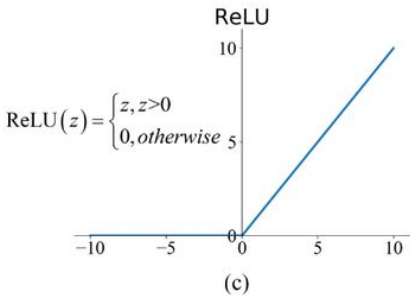


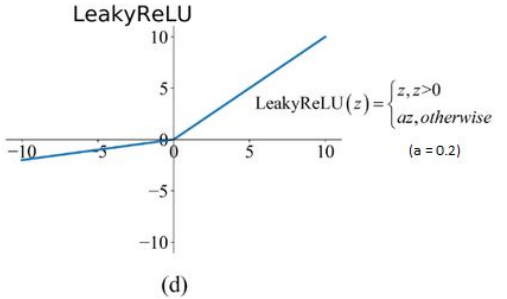
Activation Functions

Activation functions are used in each layer's calculations, represented as $f_i(w_i*z + b_i)$.
The **ReLU** function has become a standard in the field as it helps prevent the **Vanishing Gradient** problem.









How does a neural network learn?

Neural Networks (NNs) are trained using **first-order** optimization methods, such as Gradient Descent, **Adam**, and similar techniques.

We will now define the **loss function** and explain how **backpropagation** is used to efficiently compute the gradients in our network.

Loss Functions

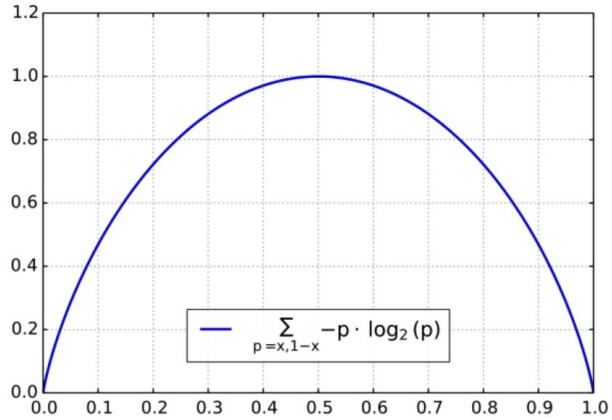
The **Mean Squared Error** is typically the most common choice of loss function for regression models, whereas **Cross Entropy Loss** is preferred for classification tasks. These measure the **error of the prediction from the ground truth.**

Mean Error Squared

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

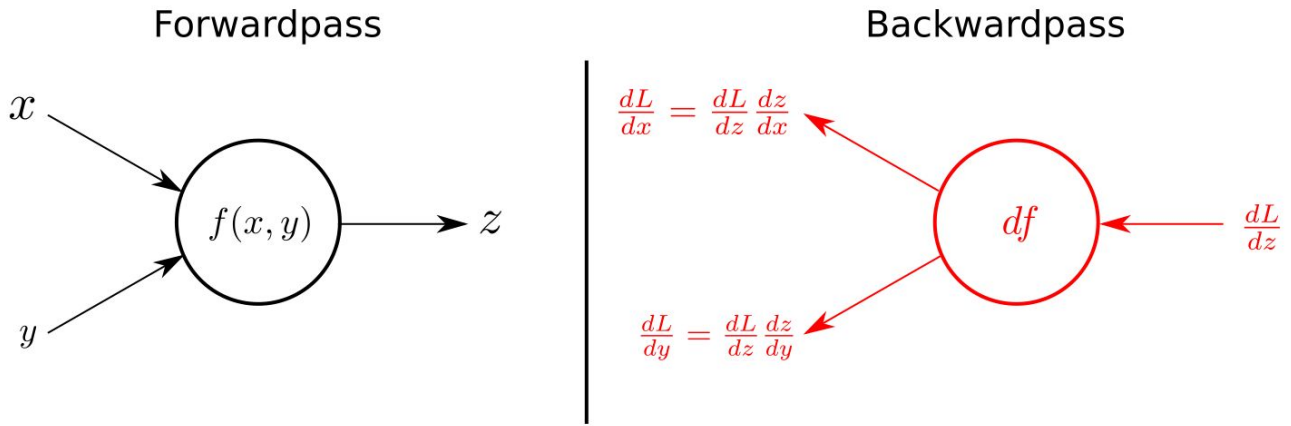
$$H(P^* \mid P) = -\sum_i \underbrace{P^*(i)}_{\text{TRUE CLASS DISTIRBUTION}} \log \underbrace{P(i)}_{\text{PREDICTED CLASS DISTIRBUTION}}$$

Two dimensional case of CE:



Backpropagation

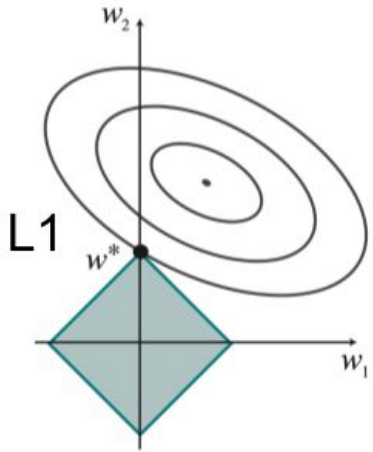
After making a prediction, we can save the data generated during the computation and use it to **calculate the gradient** with respect to the input effortlessly, starting from the gradient with respect to the output (loss function).

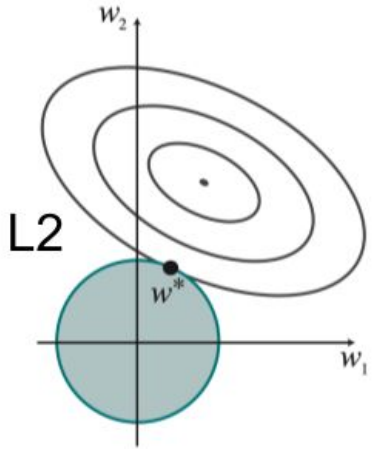


Penalty Functions

We can **penalize complex models** by incorporating a regularization term into the total cost. L1 and L2 penalty functions are the most commonly used methods for this purpose.

$$Cost(h) = EmpLoss(h) + \lambda Complexity(h)$$





Performance Metrics for Classification

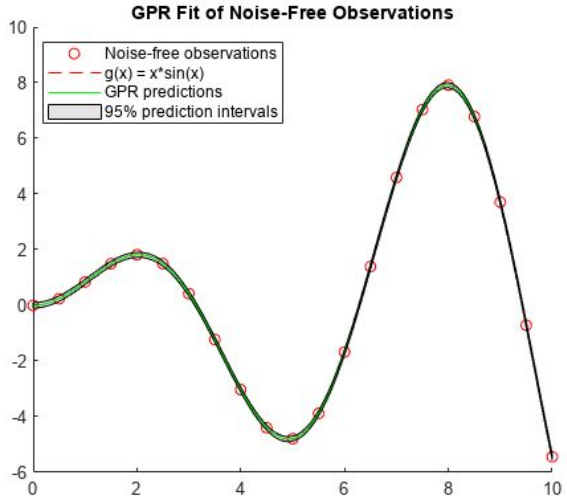
We will **evaluate the performance** of our **classifier** primarily using **overall accuracy**. Additionally, metrics such as precision, recall, and the F1 score, which is their harmonic mean, are also important for a comprehensive assessment.

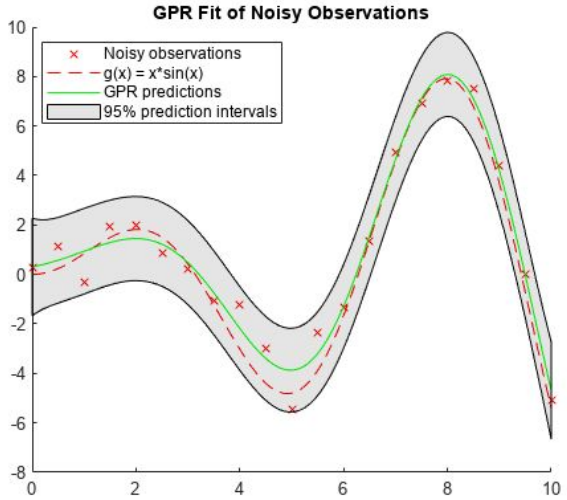
		Actual class		
		Positive	Negative	
Predicted class	Positive	**TP:** True Positive	**FP:** False Positive (Type I Error)	**Precision:** $\frac{TP}{(TP + FP)}$
	Negative	**FN:** False Negative (Type II Error)	**TN:** True Negative	**Negative Predictive Value:** $\frac{TN}{(TN+FN)}$
		Recall or Sensitivity: $\frac{TP}{(TP + FN)}$	**Specificity:** $\frac{TN}{(TN + FP)}$	**Accuracy:** $\frac{TP + TN}{(TP + TN + FP + FN)}$

Bayesian Optimization

Gaussian Process (GP) Regression

Gaussian Process (GP) Regression is a **Bayesian statistical approach** for **modeling functions**. It is based on two key concepts: the **GP posterior** on the objective function and the **Acquisition Function**.





Mean Function and Kernel

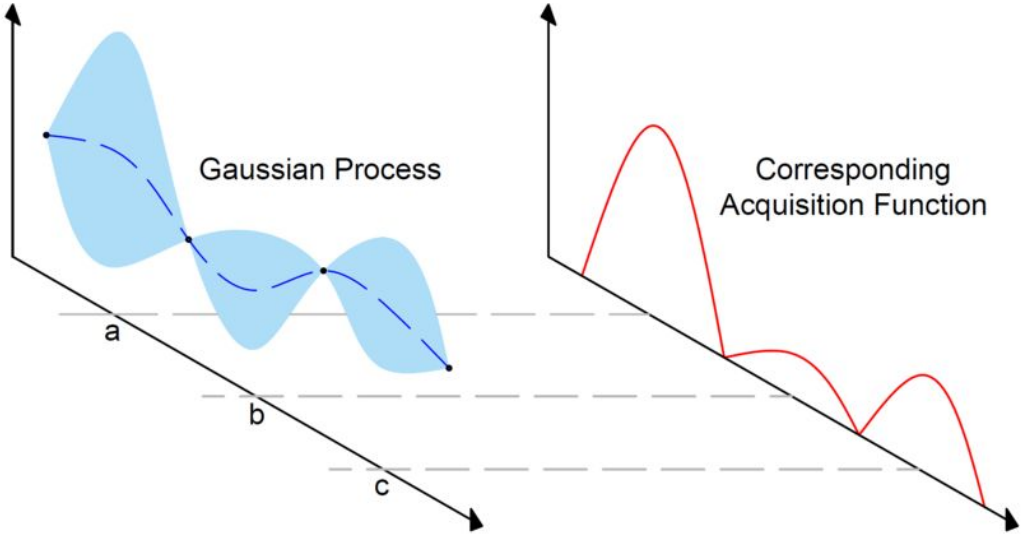
The **mean function** represents the **expected value** of the **function** we are modeling, usually $\mu_0(x) = \mu$.

While the **kernel** defines the **covariance** between **function values** at **different points** and encodes assumptions about the function's smoothness, periodicity, and other properties.

Lineal	$x^{\mathrm{T}}x_{\mathrm{j}}$	Powered	$-\left\|x-x_{\mathrm{j}}\right\|^{\beta}\ 0<\beta\leq 1$
Polynomial	$(a\times x^{T}x_{\mathrm{j}}+b)^{d}$	Log	$-log\left(1+\left\|x-x_{\mathrm{j}}\right\|^{\beta}\right)\ 0<\beta\leq 1$
RBF	$e^{(-\frac{\left\|x-x_{\mathrm{j}}\right\|^{2}}{\sigma^{2}})}$	Generalized Gaussian	$e^{-(x-x_{\mathrm{j}})^{\mathrm{T}}\mathrm{A}(x-x_{\mathrm{j}})}$ where A is a symmetric PD matrix
Sigmoid	$\tanh(\sigma x^{\mathrm{T}}x_{\mathrm{j}}+r)$	Hybrid	$e^{-\frac{\left\|x-x_{\mathrm{j}}\right\|^{2}}{\sigma^{2}}}\times(\tau+x^{\mathrm{T}}x_{\mathrm{j}})^{d}$

Acquisition Function

The acquisition function is a crucial component in GP regression. It **guides the selection of the next points for evaluation** by **balancing exploration** (searching new points) and **exploitation** (finding the best point).



Expected Improvement and UCB

Those are the most common Acquisition Functions.
The closed form of EI, under the common assumption that the prediction follows a Gaussian distribution.
q-EI is used in the case of parallel evaluations.

$$I(x) = \max(f(x) - f(x^+), 0)$$

$$\mathrm{EI}(x) = \mathbb{E}[I(x)] = \mathbb{E}[\max(f(x) - f(x^+), 0)]$$

$$\mathrm{EI}(x) = (\mu(x) - f(x^+))\Phi\left(\frac{\mu(x) - f(x^+)}{\sigma(x)}\right) + \sigma(x)\phi\left(\frac{\mu(x) - f(x^+)}{\sigma(x)}\right)$$

$$\mathrm{PI}(x) = \mathbb{P}(f(x) > f(x^+))$$

$$\mathrm{UCB}(x) = \mu(x) + \kappa\sigma(x)$$

$$\text{q-EI}(\mathbf{x}_{1:q}) = \mathbb{E}\left[\max\left(\max_{i=1,\dots,q} f(x_i) - f(x^+), 0\right)\right]$$

where $\mathbf{x}_{1:q} = \{x_1, x_2, \dots, x_q\}$ are the points to be evaluated in parallel, and $f(x^+)$ is the best observed value so far.

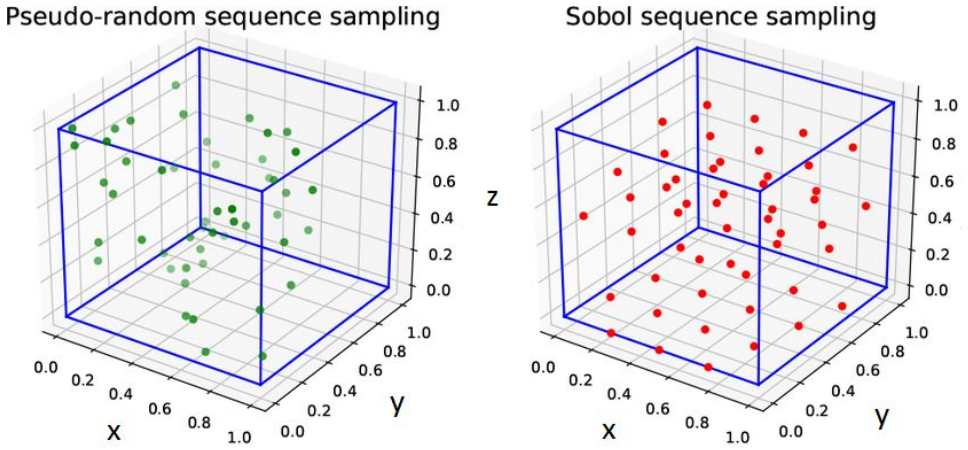
Sobol' Sequences

During the initial phase, when there are very few data points, finding the global **maximum** of the **acquisition functions** can be **challenging**. This is because the mean and variance functions **tend to flatten far from the evaluated points**.

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(x_i) = \int_D f(\tau)\, d\tau$$

to converge as fast as possible.

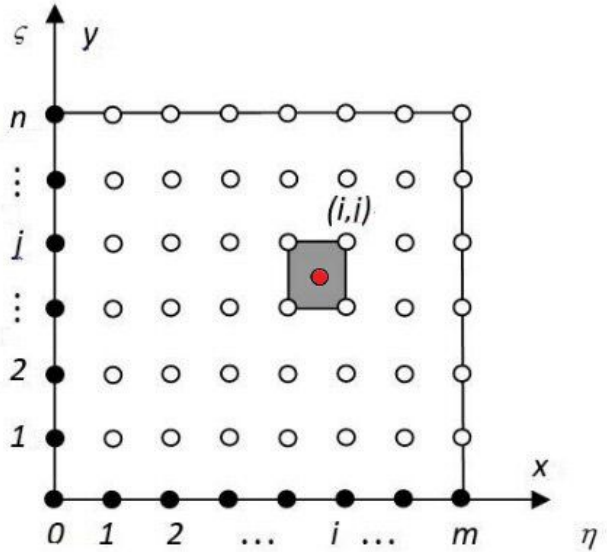
Sobol' is a **quasi-random** method that generates a **more evenly distributed** sample space compared to **deterministic grid generation.**



Integer Variables Approximations

The underlying assumption for the objective function is that it is at least continuous. This assumption guarantees the existence of a maximum due to the validity of Weierstrass Theorem.

In computer experiments we will often need to work with **discrete values**, in that case, having a **simple set** grants us **regularity conditions** that enables us to round to the **nearest integer**.



First Pseudocode

Algorithm 3: Bayesian Optimization Loop

Data: number of initial space-filling points n_0, total number of samples to produce N

Observe f at n_0 points according to a Sobol' sequence.

$n \leftarrow n_0$

while $n \leq N$ **do**

  Update the surrogate model using all available data.

  Let x_n be a maximizer of the chosen acquisition function.

  Compute the nearest integer $\hat{x}_n$ of x_n.

  Observe $y_n = f(\hat{x}_n)$.

  Increment n.

end

return The point with the largest $f(\hat{x})$ value.

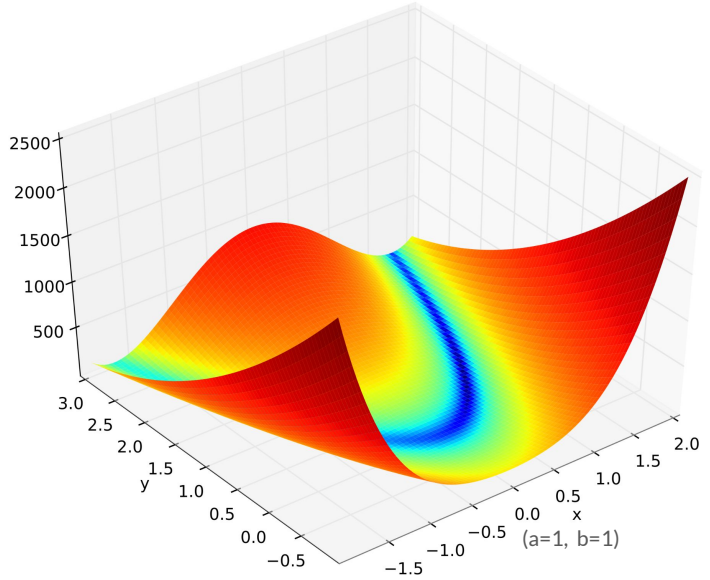
Test Function Outcomes

Rosenbrock Banana Function

we will discuss the results of the **first experiment**, which involves applying the **Ax Service API** to the **Rosenbrock** banana function. The Rosenbrock function is defined as follows:

$$f(x,y) = (a-x)^2 + b(y-x^2)^2$$

$$\nabla f(x,y) = \begin{pmatrix} -2(a-x) - 4bx(y-x^2) \\ 2b(y-x^2) \end{pmatrix}$$



Problem Formalization

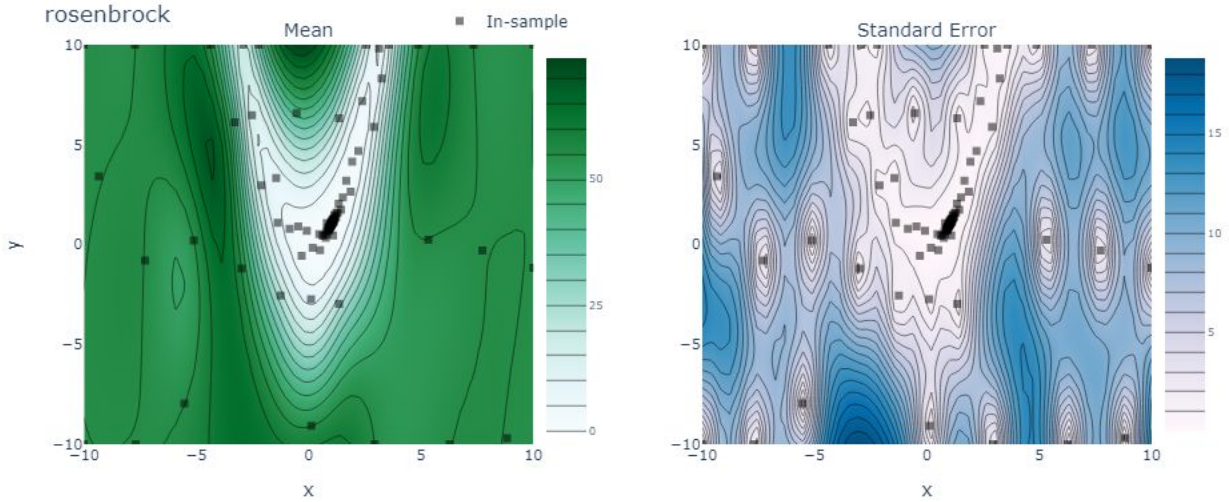
It is well-known for being difficult to optimize using **standard gradient descent methods**. In fact, even **using a random descent** direction can be more **efficient** when employing **Armijo's** method.

We run 100 trials, assign **a=1, b=1** and this as the feasible set:

$$\begin{cases} \min f(x,y) = (1-x)^2 + (y-x^2)^2 \\ x \in [-10, 10] \\ y \in [-10, 10] \end{cases}$$

Results

Without the noise assumption, the best result inferred by our algorithm occurs at the 91st trial with (x, y) = (1.00292, 0.99658)



First Observation

A smaller feasible set makes it easier to obtain good data. A useful approach could be to conduct a few initial trials, then narrow the domain around the latest best point. (Like the Hill Climbing Algorithm)

Algorithm 4: Ax Inference and Zoom

Data: a starting feasible set $\mathcal{L}$, a function $f(x)$ to minimize
$\Delta \leftarrow$ the inferred model of $f(x)$ on $\mathcal{L}$.
for $k = 0, 1, 2, ..., N$ **do**
  $\hat{x} \leftarrow$ the current best point found with Ax in Δ.
  $\mathcal{L} \leftarrow$ a smaller feasible set around $\hat{x}$.
  Δ is re-inferred using only the new currently feasible points in $\mathcal{L}$.
end
return $\hat{x}$

This technique can **easily get stuck in local minima.**

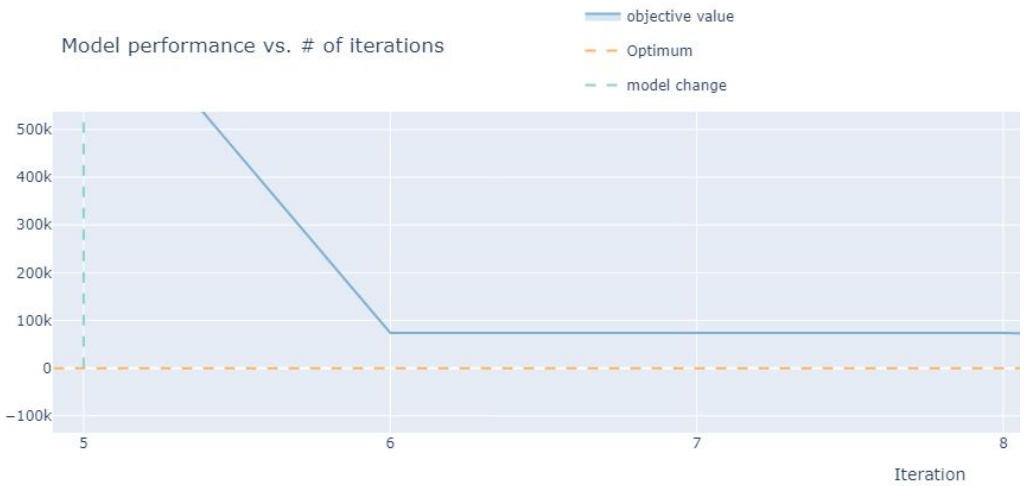
Impracticality and Noise-less Assumption.

As the **model becomes increasingly complex** with **each iteration**, it becomes more difficult for B.O. to find the optimum. In reality, **even 500 iterations can take an impractically long time to infer the model** compared to the actual function evaluation.

A **noise-less assumption can be reasonably applied** if we assume that a **script running with the same parameters** returns the **same output every time**. If this were not the case, evaluating the same point could be crucial to estimate the noise. **Ax has a built-in function to infer noisy spaces.**

Second Experiment

With a relaxed domain: $\begin{cases} x \in [-100, 100] \\ y \in [-100, 100] \end{cases}$



The **best point found** is at the **84th trial** with (x, y) = (0.81589, 0.74275), which is noticeably farther from the optimum compared to the previous results, supports our theory that **larger domains degrade performance.**

Case Study & Considerations

Hyperparameters Overview

- Number of Layers (Challenging, see next slide)
- Neurons per Layer
- Kernel Size and Stride
- Activation Functions
- Optimizer (learning rate and other parameters such as in Adam)

Number of Layers

It makes **challenging** to manage the **dependent parameters** because this variable **influences the number of other variables** in the function we are defining to optimize.

A solution could involve using **two B.O. loops**: the **first to select the number of layers**, and the **second to optimize the other parameters**.

This approach requires significantly more work and could be an extension of our project in the future.

Neurons per Layer

if n_i **is the number of neurons in layer i,** our equation would be $n_i = Kx_i$**, where** x_i **is the input value** from our Bayesian optimization loop and **K is a constant**, typically 12 or 16.

This constant K functions as a **granularity factor** and can be adjusted according to **Ax Inference and Zoom** (Alg 4).

Additionally, K prevents Ax from selecting a variation of **only 1-2 neurons per trial,** which would cause minimal variation in overall accuracy and **generate a locally flat utility function**.

Optimizer Parameters

The choice of the optimizer is also important, but **Adam** is very often the best choice.

The **learning rate α**, its **forgetting factor β_1** and **second momentum β_2** can also be included in the optimization process.

Curse of Dimensionality

Excluding the number of layers, a qualitative calculation suggests that we **have more than 20 variables to consider!**

This is **not advantageous** because **more dimensions require more trials to effectively optimize** our function.

Additionally, **more trials are needed** when using **Sobol' sequences to properly initialize the model**.

Feature Engineering

Ideally, we would have an algorithm that could **take all variables** and find the **optimal solution effortlessly**. However, this is neither easy nor fast to achieve.

We **must select which variables are important** and have a significant impact on our network's performance.

Outcome constraints can also be set up, such as an **Accuracy of a specific class**, o a **Precision score** which if under our expectations makes the trial to be considered failed.

Other discarded parameters

- Kernel Size and Stride (CNN)
  - Larger sizes may **suffer more from noise** but can capture more extensive features in the image.
- Activation Functions
  - The **nonlinearity** used in **each layer** can be varied, but **ReLU** is a widely accepted standard.

Utility Function

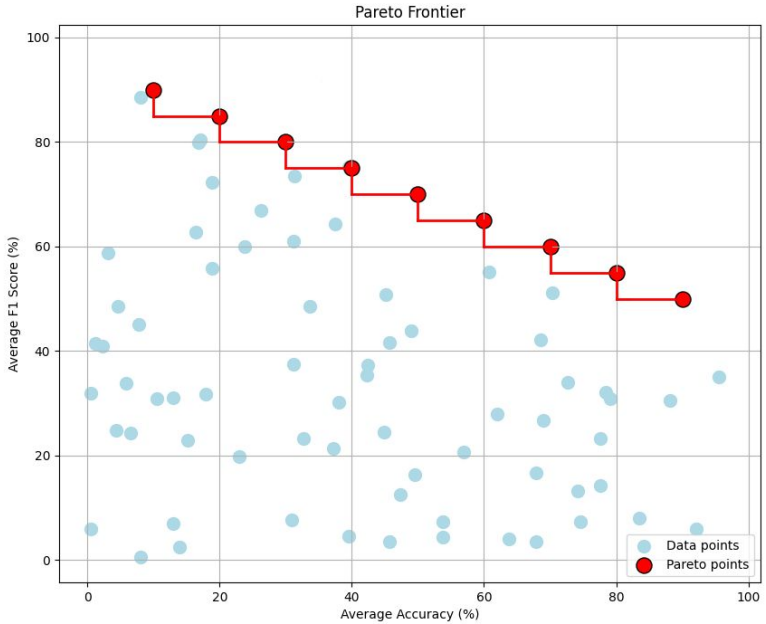
How can we **measure our network's performance?**

we have chosen to use the **average accuracy across all classes** as the utility function to maximize in our optimization loop.

Other approaches can also be implemented, such as the **F1 Score**.

Multi-Objective Optimization

Or we can move to **multi-objective optimization**
and use more than one metric, **comparing them using Pareto Frontier.**



Problem Formalization

In this project, we aim to **maximize our utility function** with respect to the **number of neurons** in each layer **ni**, where **l is the number of modifiable layers**, and the **learning rate α** of the **Adam** Optimizer.

The **utility function** is bounded so **no transformation** is needed.

$$\begin{cases} \max U(n, \alpha) \\ n = K^T x \\ x_i \in [1, a_i] \\ n \in \mathbb{N}^l \\ x \in \mathbb{N}^l \\ K \in \mathbb{N}^l \\ \alpha \in \{10^{-2}, 10^{-3}, 10^{-4}\} \end{cases}$$

$$K^T = (12, 12, ..., 12) \text{ and } \forall i, a_i = 16.$$

$$U(n, \alpha) \in [0, 100] \subset \mathbb{R}$$

Neural Network Structure

1. Convolution - Batch Normalization - ReLU
2. Convolution - Batch Normalization - ReLU
3. Max Pooling
4. Convolution - Batch Normalization - ReLU
5. Convolution - Batch Normalization - ReLU
6. Fully Connected (Linear)

Convolutional Layers:

- Input Channels
- Output Channels
- Kernel Size (dimensions of the convolution kernel)
- Stride (pixels by which the filter moves across the input)
- Padding (pixels added to the input matrix around the border)

Batch Normalization normalizes the inputs to have zero mean and unit variance.

The **max pooling** helps maintain the **robustness of feature detection** (object location in the image) and also **reduces the spatial dimensions** of the data.

First Discoveries

Our initial trial involved **manually training** our network, resulting in an average **accuracy of 73%**, with n = (12, 12, 24, 24) and α = 0.001.

In our first experiments, using a_i =8 we observed that even during the Sobol' sequence generation, a better result was achieved at the 6th trial with U6(n, α) = 79.73%.

The maximum was found at the **22nd trial with U22(n, α) = 80.85%.** The algorithm was run for 50 trials.

Unexpected Event

Starting from the 22nd trial, the Bayesian optimization algorithm **consistently decided to optimize the same point**. This can be explained by the fact that **we are dealing with an integer optimization problem**. When **relaxing to real numbers to find the maximum of our acquisition function**, we **repeatedly obtained a point near the sampled one**, which was then truncated to the same point.

23	BoTorch	48	96	96	60	10^{-4}	80.85
24	BoTorch	48	96	84	60	10^{-4}	80.38
25	BoTorch	48	84	96	72	10^{-4}	80.4
26	BoTorch	48	84	84	60	10^{-4}	80.03
27	BoTorch	60	96	96	72	10^{-4}	80.53
28	BoTorch	36	96	96	72	10^{-4}	79.86
29	BoTorch	60	96	96	48	10^{-4}	80.31
30	BoTorch	48	96	96	60	10^{-4}	80.85
31	BoTorch	48	96	96	60	10^{-4}	80.85
32	BoTorch	48	96	96	60	10^{-4}	80.85
33	BoTorch	48	96	96	60	10^{-4}	80.85

Regularization Implementation

In the next trials we subsequently increased a_i to 16.

This significantly **increased the network complexity**, resulting in **wider layers that further slowed down** the **training process**. It could be beneficial to **add regularization techniques** such as Lasso or Ridge to prevent the network from overfitting, or **add a cost penalty to the Utility Function.**

The Restart algorithm

Another useful option is to **save all experimental data and use it later** to re-infer the model, **starting another round of trials**, as shown in the example below.

Algorithm 5: Ax Restart

Data: a feasible set $\mathcal{L}$, a function $f(x)$ to minimize, a set Σ of already done trials.

$\Delta \leftarrow$ the inferred model of $f(x)$ on $\mathcal{L}$ using Σ.

for $k = 0, 1, 2, ..., N$ **do**

| Do the usual bayesian optimization loop things...

end

$\hat{x} \leftarrow$ the current best point found with Ax.

return $\hat{x}$

The Restart algorithm

Another useful option is to **save all experimental data and use it later** to re-infer the model, **starting another round of trials**, as shown in the example below.

Algorithm 5: Ax Restart

Data: a feasible set $\mathcal{L}$, a function $f(x)$ to minimize, a set Σ of already done trials.

$\Delta \leftarrow$ the inferred model of $f(x)$ on $\mathcal{L}$ using Σ.

for $k = 0, 1, 2, ..., N$ **do**

| Do the usual bayesian optimization loop things...

end

$\hat{x} \leftarrow$ the current best point found with Ax.

return $\hat{x}$

Contour Plots

The observed **concavity** suggests that we have likely identified the **maximum** of our function.

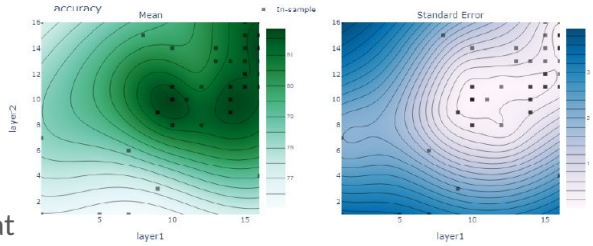


Figure 4.6. Contour Plot of x_1 and x_2

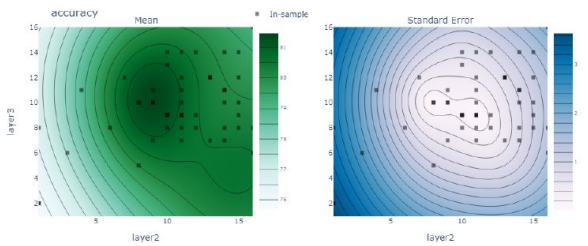


Figure 4.9. Contour Plot of x_2 and x_3

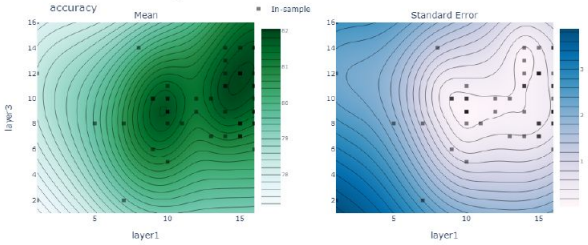


Figure 4.7. Contour Plot of x_1 and x_3

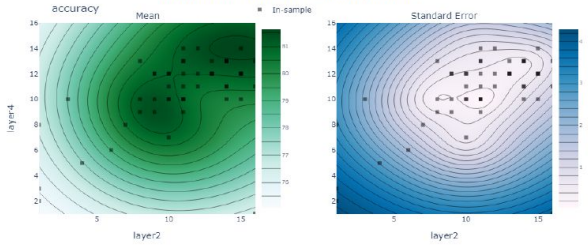


Figure 4.10. Contour Plot of x_2 and x_4

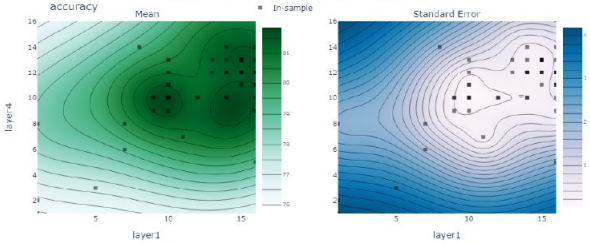


Figure 4.8. Contour Plot of x_1 and x_4

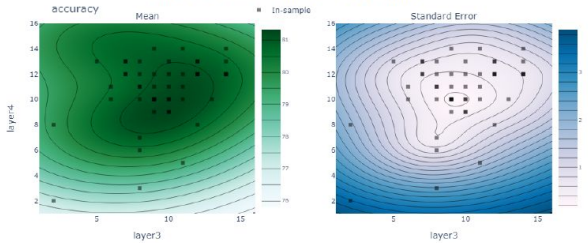
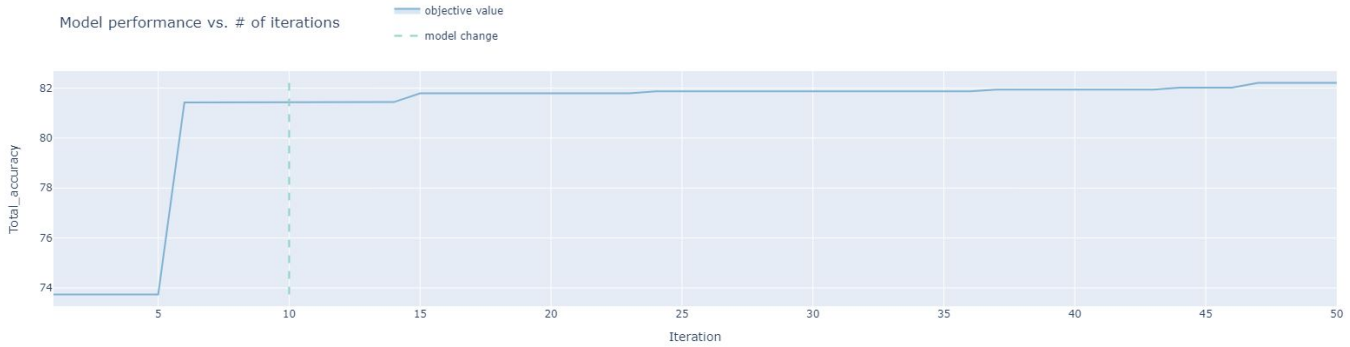


Figure 4.11. Contour Plot of x_3 and x_4

Final Results

The best results were achieved in the **46th trial**, where $U_{46}(n, \alpha) = 82.21\%$ with the input n = (192, 168, 168, 156) and α = 0.0001.

The experiment concluded in approximately 25 hours.



Conclusions

Results

This project **aimed to study Bayesian optimization** as a derivative-free optimization method within the realm of mathematical programming.

It has enhanced my understanding of **Operations Research** as a future **Computer Engineer**. It deepened my knowledge of key tools like **PyTorch**, BoTorch, and Ax, and gave me valuable **experience** in **studying** and applying concepts from **academic papers.**

This experience not only improved my technical skills but also, by **working independently**, prepared me to **approach complex research problems** with a more analytical and informed perspective.

Future Developments

Numerous improvements can be made in the future. As repeatedly mentioned during our analysis, **outcome constraints** could be **implemented**, based on time and **other metrics**, as well as **multi-objective optimization** and introducing **penalties** for overly **complex models**.

Entirely **different fields of application can be considered**, such as **chemical reactions**, where we could use a different acquisition function that allows for **both parallelism and noisy environments.**

Thanks!

Q&A time!