Assignment 4: AutoML

Ali Eren Demir

December 5, 2025

Contents

Running Instructions

- **GitHub Username:** `alieren-demir`
- **Repository URL:** `https://github.com/alieren-demir/mlds-hw4`

Environment Setup

To ensure all required libraries and dependencies are installed, please create the Conda environment from the `environment.yml` file provided in the repository.

```
# 1. Create the environment from the file
conda env create -f environment.yml
# 2. Activate the environment
conda activate myenv
```

Running the Code

Once the environment is activated, both of the parts of the homework can be run using the following commands:

```
python hw4_final.py
```

1 Introduction

Per the requirements, the data was processed manually into inputs (X) and outputs (y), filtered for digits 0-9, and split into 80% training and 20% validation sets for the tuning phase. The test set was reserved strictly for the final evaluation.

2 Part 1: Genetic Algorithm

2.1 Implementation Details

I implemented a Genetic Algorithm (GA) from scratch without using specialized packages. The implementation adheres to the following specifications:

- **Encoding:** Binary chromosome representing Batch Size (mapped linearly to [16, 1024]) and Activation Function (ReLU, Sigmoid, Tanh).
- **Selection:** Roulette Wheel selection based on fitness (Validation F1 Score).
- **Crossover:** One-point crossover applied to parent gene sequences.
- **Survivor Selection:** I implemented an age-based selection mechanism where the oldest individuals are replaced by new offspring. To ensure stability and convergence, I strictly preserved the single best individual (elitism or fitness selection) from the previous generation if the new population's best fitness dropped.

2.2 Fitness History

The population size was set to 50 with 30 generations. The plot below illustrates the Average and Highest fitness scores over generations.

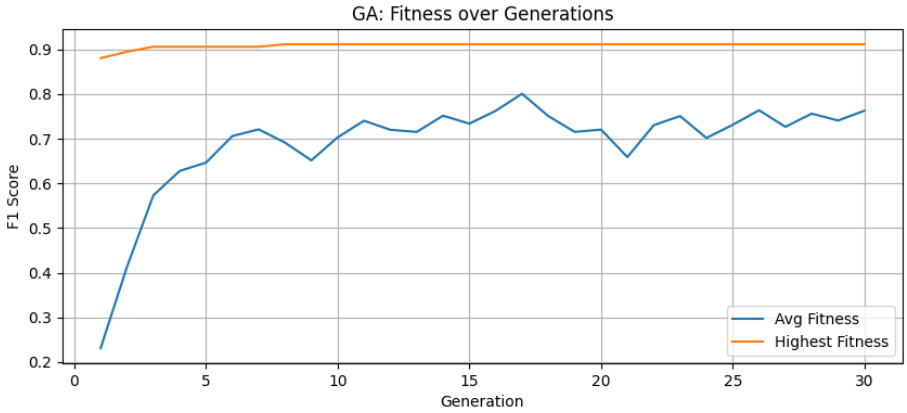

Figure 1: Average and Highest Fitness Score vs. Generation.

Analysis: As shown in Figure 1, the algorithm demonstrated healthy evolution. The average fitness (blue line) steadily increased from ≈ 0.23 to ≈ 0.76, indicating that the population successfully filtered out unstable hyperparameter configurations. The maximum fitness converged to ≈ 0.91 by generation 8 and remained stable.

2.3 Selected Hyperparameters & Final Evaluation

The vector with the highest fitness score in the last generation yielded the following hyperparameters:

- **Mini-batch Size:** 69
- **Activation Function:** Tanh

Using these selected hyperparameters, the model was retrained on the combined training and validation data for 100 epochs.

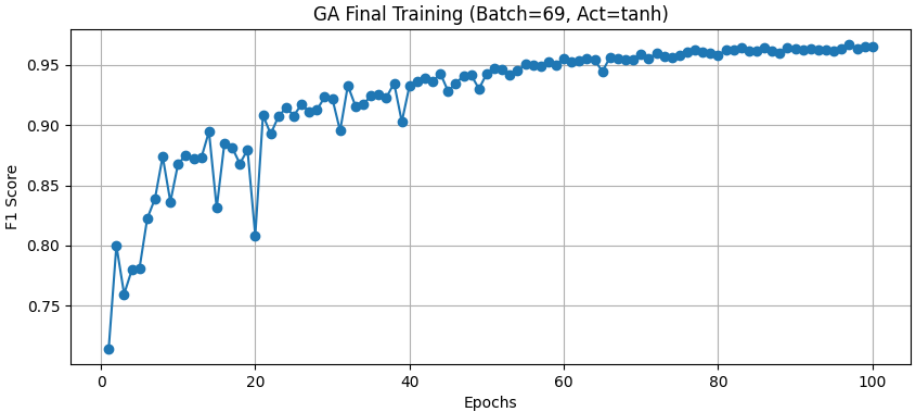

Figure 2: Training F1 Score vs. Epochs (GA Selected Parameters).

The final performance on the held-out test set is:

Test F1 Score: 0.9518

3 Part 2: Bayesian Optimization

3.1 Implementation Details

I utilized the `bayes_opt` package to tune the mini-batch size and activation function. The black-box function was defined as the validation F1 score after training for a fixed number of search epochs.

3.2 Progress Report

The optimization ran for 100 iterations (20 random initialization steps + 80 optimization steps). Below is a sample of the progress output, showing initial exploration and convergence.

Iteration	Target (Val F1)	Batch Size	Activation Code
1	0.0172	393.54	2.85 (Tanh)
2	0.0173	753.85	1.80 (Sigmoid)
6	0.9007	36.75	2.91 (Tanh)
20	0.1015	705.71	1.32 (Sigmoid)
42	0.0206	309.37	0.00 (ReLU)
65	0.9008	97.86	2.99 (Tanh)
82 (Best)	**0.9016**	**56.46**	**2.87 (Tanh)**
93	0.6174	44.27	1.06 (Sigmoid)

Table 1: Bayesian Optimization Progress Sample.

3.3 Selected Hyperparameters & Final Evaluation

The Bayesian Optimization algorithm converged to the following optimal hyperparameters:

- **Mini-batch Size:** 56
- **Activation Function:** Tanh

The model was retrained on the combined training and validation data using these parameters.

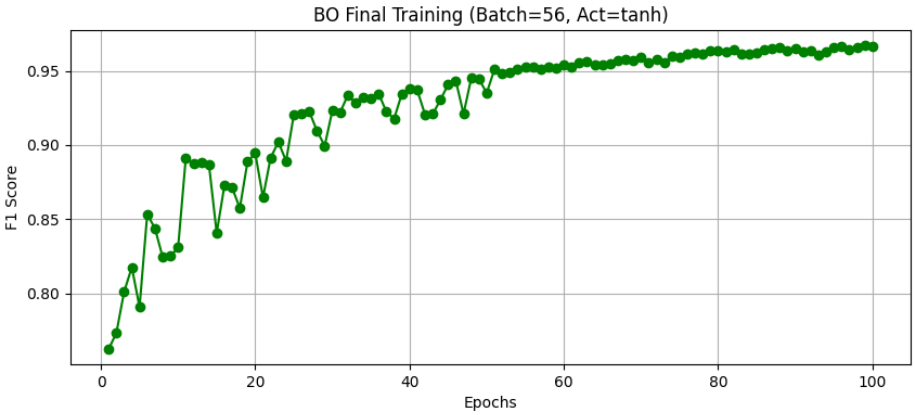

Figure 3: Training F1 Score vs. Epochs (BO Selected Parameters).

The final performance on the held-out test set is:

Test F1 Score: 0.9512

4 Comparison and Discussion

4.1 Hyperparameter Comparison

Both algorithms converged to nearly identical configurations:

- **Genetic Algorithm:** Batch 69, Tanh
- **Bayesian Optimization:** Batch 56, Tanh

This convergence suggests that for this specific architecture and dataset, a small batch size combined with the Tanh activation function represents the global optimum. It is worth noting that obtaining high performance with Tanh required the implementation of a Learning Rate Scheduler and Linear Scaling Rule to prevent gradient instability during the final 100-epoch training phase.

4.2 Pros and Cons

Based on the execution of both methods, I observed the following trade-offs:

Genetic Algorithm (GA):

- **Pros:** Highly robust to instability. The population-based approach naturally filtered out "lethal" hyperparameter combinations (e.g., those causing exploding gradients), as indicated by the steady rise in average fitness. It handles discrete variables (like activation function indices) naturally.
- **Cons:** Computationally expensive. The GA required 1,500 total model evaluations (50 pop $\times$ 30 gens) to reach the same conclusion that Bayesian Optimization reached in 100 iterations.

Bayesian Optimization (BO):

- **Pros:** extremely sample-efficient. It identified high-performing regions (F1 $>$ 0.85) relatively early in the search process (around iteration 60).
- **Cons:** Struggles with discrete/categorical variables. Because the activation function was encoded continuously, the algorithm wasted iterations exploring the "dead space" between integers (e.g., trying to evaluate an activation code of 1.5), which resulted in several failed evaluations with near-zero F1 scores.