Large Language Models as Hyperparameter Optimizers

Stephanie Zhou, Jeena Mahajan, Dredheza Gashi Baraliu

MATH 467 – Professor Alcala

Motivation: LLMs & Hyperparameter Tuning

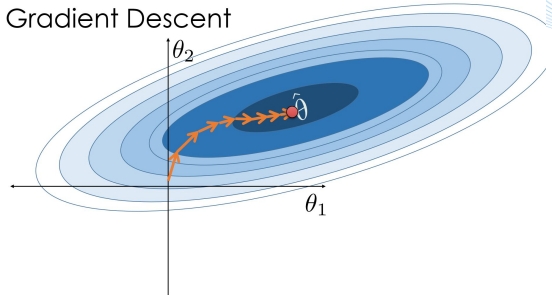
- Hyperparameter tuning: critical for optimization performance
  - Bad parameters = bad performance
    - Wasted iterations
    - Divergence
- Traditional search methods like grid search and random search
  - Costly
  - Can be inefficient
- LLMs: have learned patterns across massive amounts of data
- **Opportunity**: Use LLMs to suggest hyperparameter values for optimization problems

Research Question: Can LLM's suggest effective hyperparameters for optimization algorithms, outperforming traditional search methods?

Gradient Descent

- Purpose: the optimization algorithm we'll use to minimize the benchmark function
- Core Idea:
  - Goal: minimize the function *f(x)*
  - Gradient: direction of steepest increase
    - To minimize, move in the opposite direction
- Iterative update rule: move a small amount "α" downhill

$$x_{\{k+1\}} = x_k - \alpha \nabla f(x_k)$$



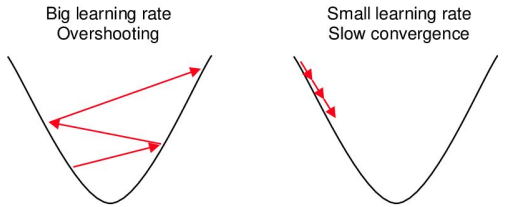
- Intuition for Gradient Descent = repeatedly taking small downhill steps until you reach the "valley"

Gradient Descent: Hyperparameters

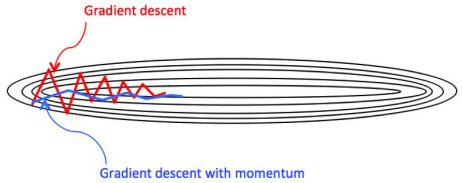
$$v_{t+1} = \beta\, v_t - \alpha \nabla f(x_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

- Learning rate (α):
  - Too small → very slow
  - Too large → overshoot or diverge
  - "Just right" → fast and stable convergence
  - Intuition: how "big" of a step you take downhill

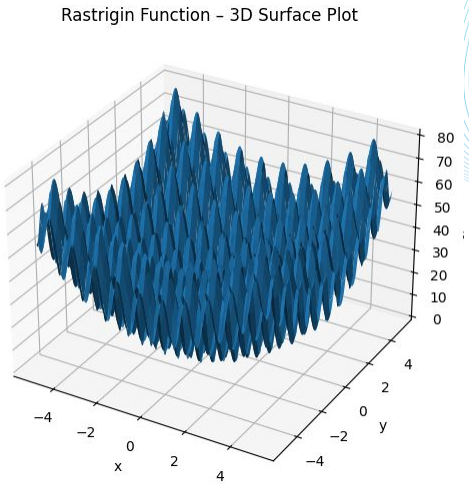


- Momentum: vector **v**
  - Adds memory of past steps to smooth the path.
    - **v** "remembers" past gradients
  - Helps push through small bumps or noisy slopes.
  - Prevents getting stuck in shallow valleys.



Benchmark Functions

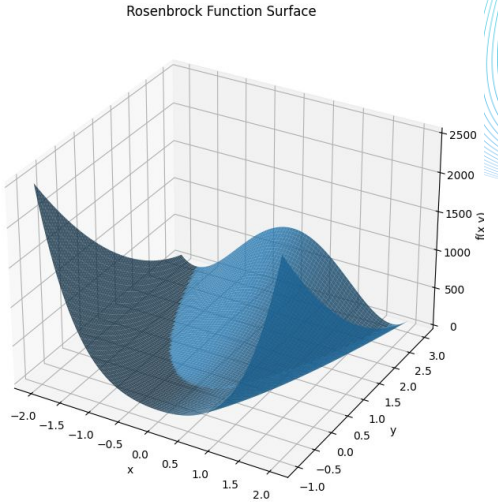
- Rastrigin Function
  - $f(x, y) = 20 + (x^2 - 10\cos 2\pi x) + (y^2 - 10\cos 2\pi y)$, global min at (0, 0), f = 0
  - Highly multi-modal, many shallow local minima
    => easy to get trapped
  - In our experiments:
    GD in R^2, x_0 = (-1.5, 2)
  - Used to test whether our learning-rate choices keep gradient descent moving smoothly on a bumpy surface



Benchmark Functions

- Rosenbrock Function
  - $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$, global min at (1, 1), f = 0
  - Narrow, curved valley => step size-sensitive
  - In our experiments:

    GD in R^2, x_0 = (-1.5, 2) [randomly sampled]
  - Used to test whether hyperparameters allow fast, stable convergence along the valley



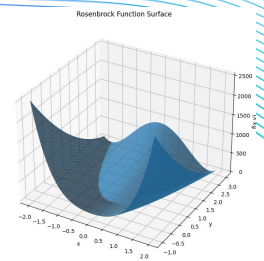
Methods

1. Grid Search
   - Test preset (learning rate, momentum) pairs
   - Run gradient descent with momentum; keep the lowest final value
   - LR: [1e-4, 1e-3, 1e-2, 1e-1]; Momentum: [0.0, 0.5, 0.9]
2. Random Search
   - Sample 4 random (lr, momentum) pairs
   - Run gradient descent for each; pick the best performer
   - LR: [6.2e-04 → 1.6e-03]; Momentum: [0.431 → 0.539]
3. LLM-Guided Optimization
   - LLM receives function info + valid lr/m ranges
   - Returns 3 candidate (lr, momentum) pairs
   - Run gradient descent for each; choose the lowest final value
   - LR: [1.0e-04 → 1.0e-03]; Momentum: [0.445 → 0.850]

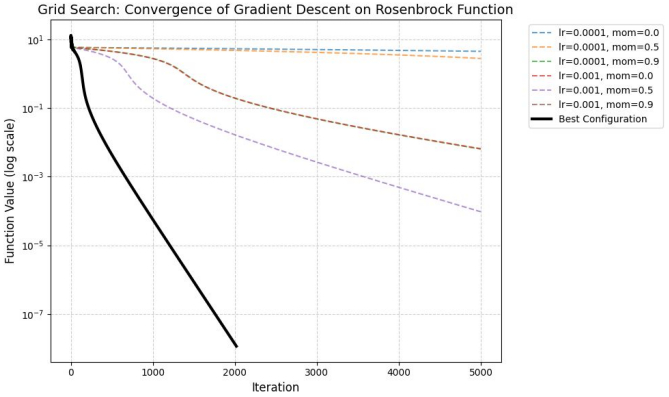
AI Models Used

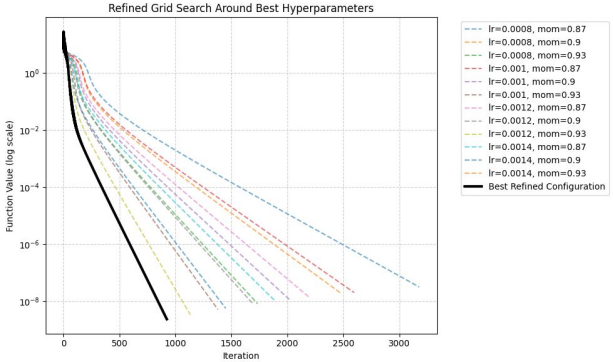
1. OpenAI GPT-4o
    - Higher-capability model
2. OpenAI GPT-4o-mini
    - Lighter, faster, cheaper variant

Results: Grid Search (Rosenbrock)

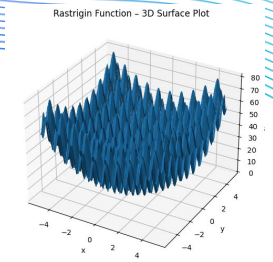


- "Best" combination was lr =0.001, momentum = 0.9
  - Best: lowest function value (2.387590e-09) in smallest number of iterations: **2020**
- Observations:
  - Very small learning rate: slow convergence, takes up all 5000 iterations & doesn't reach lowest value
  - High learning rates: diverge
  - Momentum + Average Learning Rate:
    - Builds up velocity across the curved valley, but learning rate is small enough to not overshoot the valley
- Performed a "refined" grid search around this optimal to see if we could achieve a better optimal hyperparameter combination
  - Result: Learning Rate: 0.0014, Momentum: 0.93; Iterations: **925**

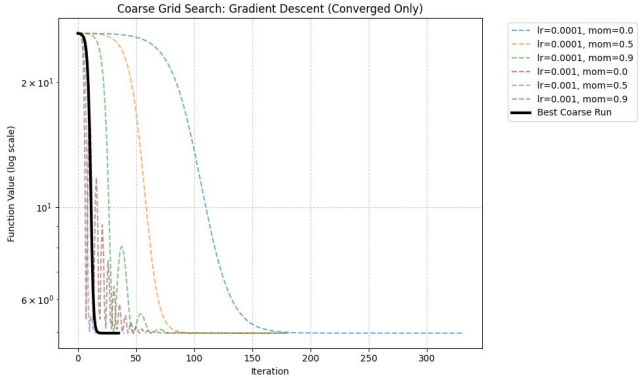


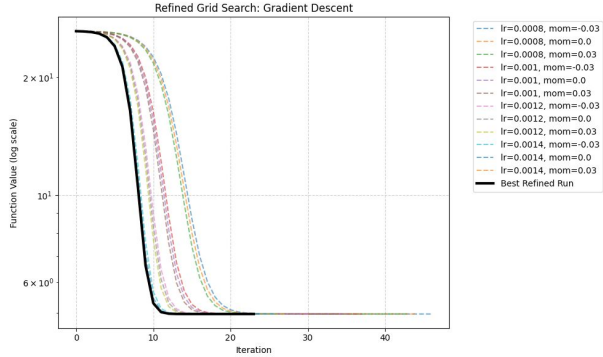


Results: Grid Search (Rastrigin)



- "Best" combination was lr =0.001, momentum = 0
  - Best: reaches local minimum (4.974790) in smallest number of iterations: **36**
- Observations:
  - Very small learning rate: slow convergence, takes up all 5000 iterations & doesn't reach lowest value
  - High learning rates: diverge -> avoid the many "small basins" due to large jumps
  - Momentum: hurts in this case
    - Without momentum, GD moves cautiously and avoids oscillations in the rugged landscape
    - With momentum, GD overshoots small basins and gets stuck for longer.
- Performed a "refined" grid search around this optimal to see if we could achieve a better optimal hyperparameter combination
  - Result: Learning Rate: 0.0014, Momentum: 0.03; Iterations: **24**





Results: Random Search (Rosenbrock)

- Performance Summary
  - Highly unstable across seeds
  - Only seed 0 produced a near-optimal value
  - Seeds 1 and 2 failed to converge well
  - All runs hit the full 2000 iterations

Seed	Best LR	Momentum	Final Value	Iterations
0	3.6001e-04	0.861	0.001826	2000
1	2.5639e-05	0.869	0.815882	2000
2	5.1045e-04	0.489	0.021873	2000

Results: Random Search (Rastrigin)

- Performance Summary
  - Very unstable across sampled hyperparameters
  - Fast convergence: ~44-51 iterations
  - Seed 0 performs the best: reaches global minimum

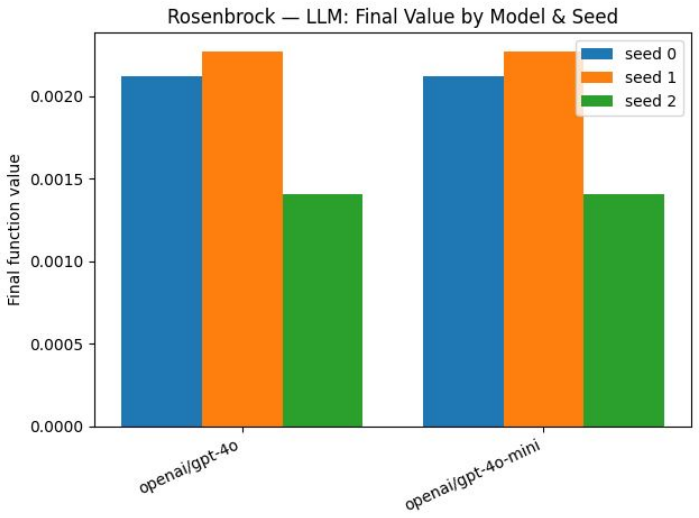
Seed	Best LR	Momentum	Final Value	Iterations
0	1.6052e-03	0.539	1.6052e-03	51
1	6.2163e-04	0.533	0.994959	51
2	1.2570e-03	0.431	3.97983	44

Results: LLMs (Rosenbrock)

- Performance Summary
  - GPT-4o & GPT-4o-mini are extremely stable
    - LR: 5 x 10^{-4}

      Momentum = 0.80
    - Final values consistently between

      0.00141-0.00227
  - All runs use the full budget (slow convergence)



Results: LLMs (Rosenbrock)

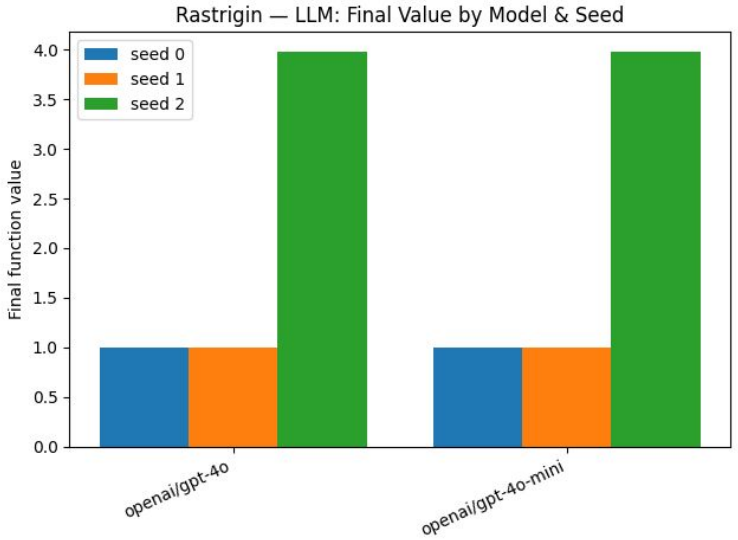
Seed	Model	LR	Momentum	Final Value	Iterations
0	GPT-4o	5.0e-04	0.800	0.002119	2000
0	GPT-4o-mini	5.0e-04	0.800	0.002119	2000
1	GPT-4o	5.0e-04	0.800	0.002272	2000
1	GPT-4o-mini	5.0e-04	0.800	0.002272	2000
2	GPT-4o	5.0e-04	0.800	0.001407	2000
2	GPT-4o-mini	5.0e-04	0.800	0.001407	2000

Results: LLMs (Rastrigin)

- Performance Summary
  - Seeds 0 and 1 reached the same final value ~0.994959 (~100 iterations)
  - Consistent across seeds 0 and 1, but differs for seed 2
- GPT-4o:

  LR = 5e-4 and 8e-4

  Momentum = 0.80 and 0.85.
- GPT-4o-mini: LR = 1e-4, Momentum = 0.70



Results: LLMs (Rastrigin)

Seed	Model	LR	Momentum	Final Value	Iterations
0	GPT-4o	8.0e-04	0.850	0.994959	151
0	GPT-4o-mini	1.0e-04	0.700	0.994959	98
1	GPT-4o	5.0e-04	0.800	0.994959	101
1	GPT-4o-mini	1.0e-04	0.700	0.994959	101
2	GPT-4o	8.0e-04	0.850	3.97983	151
2	GPT-4o-mini	1.0e-04	0.700	3.97983	100

Future Experimentation

- Extend experiments to additional benchmark functions
  - Test on functions with different geometries
  - Verifies whether the LLM generalizes beyond the current benchmark functions
- Evaluate alternative LLMs and use prompt engineering
  - Compare different models
  - Vary prompt structure, details and constraints to optimize the prompt itself
  - Determine whether performance reliant on the prompt

Conclusion

- **Rosenbrock**: Grid Search did the best; **Rastrigin**: Random Search did the best
  - Random Search is stochastic: might miss the “sweet” spot for the valley for Rosenbrock, but can “luck” into hyperparameters that escape local minima for Rastrigin
- LLMs produced reasonable, conservative learning rates & momentum values
  - Can be promising in higher-dimensional spaces where we need guidance without expensive computation
- On these benchmarks, simple baselines were already very strong
- GPT-4o-mini was often the most competitive LLM
- This sets up a foundation for more complex, realistic experiments

Thank you!