Advanced Topic A12: *Bayesian Optimisation*

Data Science with Python – BSc Advanced Lectures

Joerg Osterrieder

10 Minutes

Motivation: Optimising Expensive Black-Box Functions

Some objective functions are costly to evaluate – grid search is wasteful

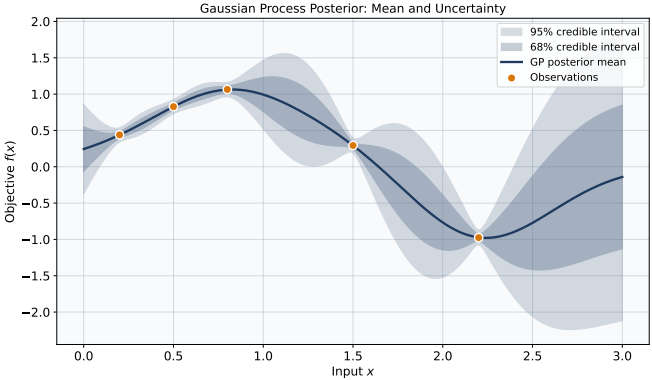
- Hyperparameter tuning: each evaluation trains a full model (minutes to hours)
- Portfolio strategy optimisation: each backtest runs a simulation over years of data
- Drug discovery, materials science: each experiment is a physical synthesis
- Grid search wastes budget on unpromising regions; random search ignores past evaluations
- **Bayesian optimisation (BO):** use all previous evaluations to choose the most informative next point; typically finds good solutions in 10–50 evaluations

BO is the standard for hyperparameter tuning when each evaluation costs more than a second

The Surrogate Model

A cheap probabilistic model that approximates the expensive objective

- Maintain a **surrogate model** $p(f|\mathcal{D})$: a posterior distribution over the objective function given all observations so far
- At each unobserved point x: the surrogate gives a mean $\mu(x)$ and uncertainty $\sigma(x)$
- After a new evaluation (x, y): update the posterior – narrower uncertainty near x
- The surrogate is fit to all past data; evaluating the surrogate is cheap (milliseconds)
- Most common surrogate: Gaussian Process (GP), though random forests and neural networks are used



The GP posterior trades off mean prediction and uncertainty, the key quantity for BO

Gaussian Processes as Surrogates

A distribution over functions with a tractable posterior

- GP prior: $f \sim \mathcal{GP}(m(x), k(x, x'))$ where k is a kernel (covariance function)
- **RBF kernel:** $k(x, x') = \sigma^2 \exp\!\left(-\|x - x'\|^2 / 2\ell^2\right)$; smooth functions; ℓ controls length scale
- GP posterior given n observations: analytically tractable Gaussian with $\mu_n(x) = k_n(x)^\top (K_n + \sigma_\varepsilon^2 I)^{-1} y_n$
- Uncertainty estimate $\sigma_n(x)$ is exact (no approximation needed for small n)
- Limitation: GP inference is $O(n^3)$; practical for $n < 1000$ evaluations

GP BO is optimal in expectation under certain regularity conditions (Srinivas et al. 2010)

Kernel Selection and Hyperparameter Tuning

The kernel encodes prior beliefs about smoothness and scale

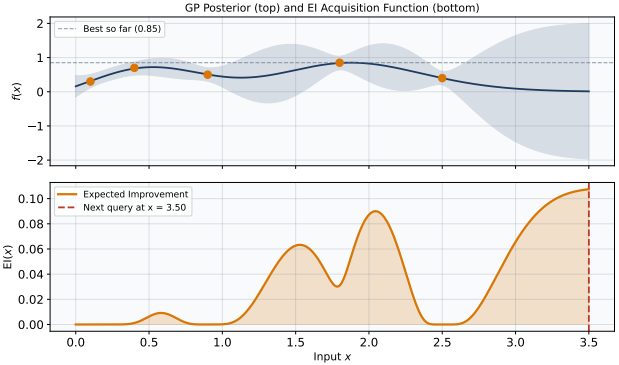
- **RBF kernel**: assumes infinitely differentiable, globally smooth functions; `length_scale` controls how quickly correlation decays with distance
- **Matern-5/2**: assumes twice-differentiable; better for non-smooth objectives common in practice; the default in BoTorch and scikit-optimize
- **Periodic kernel**: for objectives with known periodicity (seasonal financial patterns, cyclical hyperparameters)
- **Composite kernels**: sum or product of base kernels captures multi-scale structure in complex objectives
- Kernel hyperparameters (length scale, output variance) are fit by maximising the log marginal likelihood; **ARD** (Automatic Relevance Determination) assigns per-dimension length scales, identifying irrelevant input dimensions

Matern-5/2 is the safe default; use ARD when some hyperparameters are likely irrelevant

Acquisition Functions

Balancing exploration (uncertainty) and exploitation (mean)

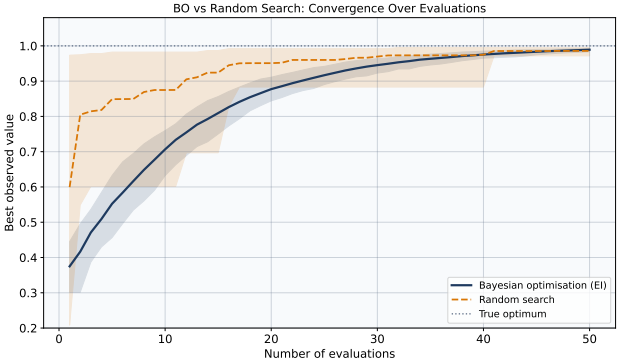
- The acquisition function $\alpha(x)$ scores each candidate point based on the surrogate
- Maximise $\alpha(x)$ to pick the next evaluation point – cheap optimisation on the surrogate
- **Expected Improvement (EI):** $\alpha_{\text{EI}}(x) = \mathbb{E}\big[\max(f(x) - f^*, 0)\big]$; closed-form for GPs; balances exploration and exploitation automatically
- **Upper Confidence Bound (UCB):** $\alpha_{\text{UCB}}(x) = \mu(x) + \kappa\,\sigma(x)$; κ controls the trade-off explicitly
- **Probability of Improvement (PI):** $\mathbb{P}(f(x) > f^* + \xi)$; greedy, less diverse



The BO Loop

Iterative sample-update-maximise cycle

1. Initialise with n_0 random evaluations (Latin hypercube or Sobol sequence)
2. Fit the GP surrogate to all observations $\mathcal{D} = \{(x_i, y_i)\}$
3. Maximise the acquisition function to find the next candidate: $x_{n+1} = \arg\max_x \alpha(x)$
4. Evaluate the expensive objective: $y_{n+1} = f(x_{n+1})$
5. Add (x_{n+1}, y_{n+1}) to $\mathcal{D}$ and repeat from step 2



BO converges in $O(\log n)$ evaluations to within ε of the optimum (under GP assumptions)

Batch Bayesian Optimisation

Selecting $q > 1$ candidates simultaneously for parallel workers

- Sequential EI wastes idle compute: when 16 backtest workers are available, picking one point at a time leaves 15 workers idle
- **Batch EI (qEI)**: the expected improvement of a batch $\{x_1, \ldots, x_q\}$ jointly; analytically intractable but estimated efficiently via quasi-Monte Carlo (BoTorch)
- **Local Penalisation** (Gonzalez et al. 2016): penalise the acquisition function near already-selected batch members; fast but approximate
- **Thompson Sampling**: sample one GP posterior function per worker; each worker selects the maximum of its sample; trivially parallel, near-optimal regret
- Finance: batch BO allocates overnight backtest slots to maximally diverse parameter regions, extracting the most information from each compute budget

Batch EI in BoTorch uses quasi-Monte Carlo sampling for efficient qEI optimisation

High-Dimensional Challenges

GP-based BO struggles beyond 20 dimensions

- GP kernel assumes smoothness over the full input space: with $d > 20$ dimensions, the space is too large for a GP to explore efficiently
- Volume of the search space grows exponentially: the n_0 initial points become infinitesimally sparse
- Solutions: **REMBO** (project to a random low-dimensional subspace); **SAASBO** (sparse axis-aligned subspace; find the relevant dimensions automatically); **TuRBO** (trust regions that focus search locally)
- **Random forests as surrogate** (SMAC, used by AutoML frameworks): better scaling to high dimensions and categorical variables than GPs

GP BO is best for $d < 20$; use TuRBO or SMAC for larger hyperparameter spaces

Multi-Fidelity and Asynchronous BO

Efficient BO when evaluations have different costs

- **Multi-fidelity BO**: evaluate cheap approximations first (short training run on 10% of data); use them to guide expensive evaluations (full training)
- **Hyperband**: successive halving based on early-stopping performance; combines with BO in BOHB (Falkner et al. 2018)
- **Asynchronous BO**: send multiple candidates to parallel workers; use GP predictions (with "fantasy" evaluations) for running jobs; batch EI or Local Penalisation
- Finance: run 16 parallel backtests; BO picks the next 16 parameter sets that are maximally diverse relative to running jobs

BOHB is the most competitive AutoML method: combines BO guidance with Hyperband efficiency

Multi-Objective Bayesian Optimisation

Optimising several conflicting objectives simultaneously

- Single-objective BO finds one point; multi-objective BO finds the **Pareto front**: the set of solutions where no objective can improve without another worsening
- Finance example: maximise Sharpe ratio AND minimise maximum drawdown – no single point dominates on both axes simultaneously
- **EHVI (Expected Hypervolume Improvement)**: acquisition function that selects the next point maximising the expected increase in Pareto front hypervolume; supported natively in BoTorch
- **Scalarisation**: convert to single-objective by weighted sum; simple but sensitive to weight choice and misses concave Pareto regions
- **NSGA-II**: evolutionary multi-objective alternative; scales better to 4+ objectives but ignores past evaluation information

EHVI is the state-of-the-art for 2–4 objectives; scalarisation works for simple two-objective tradeoffs

Limitations of Bayesian Optimisation

BO is powerful but not universally applicable

- GP cubic scaling: impractical for $n > 1000$ sequential evaluations
- Maximising the acquisition function is itself a non-trivial optimisation ($\arg\max_x \alpha(x)$ is non-convex; use multi-restart L-BFGS-B or evolutionary search)
- BO assumes a stationary GP prior: does not handle non-stationary or multimodal objectives well
- Strongly dependent on the choice of kernel and prior: wrong kernel = poor exploration
- For very cheap evaluations (milliseconds): random search or grid search are more practical

BO shines for 10–1000 evaluations of expensive functions; below 10, random search may win

Constrained Bayesian Optimisation

Optimising under black-box feasibility constraints

- Hard constraints: some parameter regions are infeasible (strategy parameters causing margin calls; model configurations violating latency budgets; leverage ratios exceeding regulatory limits)
- **Constrained EI (cEI):** multiply EI by the probability of constraint satisfaction: $\alpha_{\text{cEI}}(x) = \text{EI}(x) \cdot \prod_j \mathbb{P}(c_j(x) \leq 0)$
- Each constraint c_j is modelled with its own GP; feasibility probability is the GP CDF evaluated at zero
- **SafeOpt** (Sui et al. 2015): guarantees every evaluation falls within a safe region; suitable for live systems where infeasible evaluations carry real cost
- BoTorch supports constrained BO: pass a `constraints` argument to `ExpectedImprovement`

Constrained BO is essential when evaluating an infeasible point has real cost, risk, or regulatory consequence

Finance Application: Trading Strategy Hyperparameter Tuning

Optimising backtest performance without overfitting

- A momentum strategy has 5 hyperparameters: lookback, holding period, entry/exit thresholds, stop-loss
- Each full backtest on 10 years of tick data takes 3 minutes
- Grid search over 5 parameters at 5 levels each: $5^5 = 3125$ evaluations $=$ 156 hours
- BO with Expected Improvement: finds near-optimal parameters in 50–80 evaluations (2.5–4 hours)
- Caution: BO maximises *in-sample* backtest return – validate on out-of-sample before live trading

BO for strategy tuning: same number of backtests, much better parameter set

Finance Application: Portfolio Weight Optimisation

Tuning factor model weights with expensive risk evaluation

- A multi-factor equity portfolio has 8 factor weights; risk evaluation (CVaR on 5 years of daily returns with 10,000 Monte Carlo scenarios) takes 4 minutes per configuration
- Grid search at 3 levels per factor: $3^8 = 6561$ evaluations $=$ 18 days compute; random search wastes budget in low-Sharpe tail regions
- BO with constrained EI: maximise Sharpe subject to CVaR $\leq 3\%$; finds a near-optimal feasible allocation in 40–60 evaluations (2.5–4 hours)
- Multi-objective variant: map the full Pareto front of Sharpe vs. CVaR in 80 evaluations using EHVI; reveals the efficient frontier of risk-return tradeoffs
- Caution: BO maximises in-sample backtest Sharpe – always validate on out-of-sample data before live deployment

BO for portfolio weights: same principle as strategy tuning, multi-objective EHVI reveals the full efficient frontier

Code Pointer

Bayesian optimisation in Python

- **Ax / BoTorch** (Meta): production-grade BO; GPU-accelerated GP; supports multi-fidelity, constrained, and multi-objective BO; `pip install ax-platform`
- **Optuna**: Bayesian and TPE-based hyperparameter search; excellent integration with PyTorch, scikit-learn; `study.optimize(objective, n_trials=50)`
- **scikit-optimize**: lightweight; `gp_minimize(f, space, n_calls=50)`
- **SMAC3**: random forest surrogate; excellent for categorical + mixed spaces; used by AutoML frameworks
- Finance: wrap any backtest as a Python function; call `gp_minimize` or Optuna

Optuna is the entry point for most practitioners; BoTorch for research-level customisation

Practical Tips and Common Pitfalls

What to do – and what to avoid – when applying BO

- **Always warm-start**: use 5–10 random evaluations (Sobol sequence) before the first GP fit; a GP with 1–2 observations is unreliable
- **Normalise the objective**: scale outputs to $[0, 1]$ or standardise; GP kernels assume a stationary prior; unnormalised objectives degrade acquisition function quality
- **Log-transform parameters**: learning rate and regularisation span orders of magnitude; optimise log(lr) not lr; transforms the search space into a roughly uniform scale
- **Budget the acquisition optimiser**: maximising EI is itself a multi-restart non-convex problem; use 10–20 random restarts of L-BFGS-B to avoid local maxima
- **Watch for noise**: BO assumes a noise-free or low-noise objective; noisy backtests (stochastic simulations) require a noise-aware GP model with an explicit noise hyperparameter

BO pitfalls: skipping warm-start, unnormalised objectives, and deterministic acquisition restarts are the top three failures

Summary and Open Questions

Three things to remember

- BO uses a GP surrogate to model the objective and an acquisition function (EI, UCB) to pick the next evaluation; the loop is: fit GP, maximise acquisition, evaluate, update
- BO outperforms grid/random search by a factor of 10–100x in evaluation count for expensive objectives; practical for $d < 20$ dimensions and $n < 1000$ evaluations
- Finance: BO is the right tool for strategy parameter search, model hyperparameter tuning, and portfolio optimisation when each evaluation runs a full backtest

Open questions:

1. How do we apply BO to combinatorial or graph-structured spaces (architecture search)?
2. Can BO be used for online strategy adaptation where the objective is non-stationary?
3. How should we handle constraints (no drawdown $> 10\%$) in the acquisition function?

Further reading: Brochu et al. (2010) BO tutorial; Srinivas et al. (2010) GP-UCB regret bounds