BOAT: Building Auto–Tuner with Structured Bayesian Optimization

Author: Valentin Dalibard; Presenter: Xavier Chen (zc344)

November 13, 2024

Outline

Background: Bayesian Optimisation

Structured Bayesian Optimisation

Results & Evaluation

Conclusion

Related Works

Background: Bayesian Optimisation

Bayesian Optimisation

Motivation

Bayesian Optimisation

Motivation

- Black box function

Bayesian Optimisation

Motivation

- Black box function
- Find a minimum/maximum/optimal/satisfactory point

Bayesian Optimisation

Motivation

- Black box function
- Find a minimum/maximum/optimal/satisfactory point
- Expensive to probe

Bayesian Optimisation

Motivation

- Black box function
- Find a minimum/maximum/optimal/satisfactory point
- Expensive to probe
- Take a few samples until we are confident that we have found (close to) optimal solutions
- If only we have can represent just about *any* function $\cdots$

Gaussian Process

Think of a Gaussian Process as a *distribution of functions* $f : \mathcal{X} \to \mathbb{R}$.

Gaussian Process

Think of a Gaussian Process as a *distribution of functions* $f : \mathcal{X} \to \mathbb{R}$.

The input domain $\mathcal{X}$ can be just about anything. In most application domains it is some $\mathbb{R}^d$

Gaussian Process

Think of a Gaussian Process as a *distribution of functions* $f : \mathcal{X} \to \mathbb{R}$.

The input domain $\mathcal{X}$ can be just about anything. In most application domains it is some $\mathbb{R}^d$

By convention, we use the following notation to denote a Gaussian Process

$$\mathcal{GP}(m(\cdot), k(\cdot, \cdot))$$

Gaussian Process

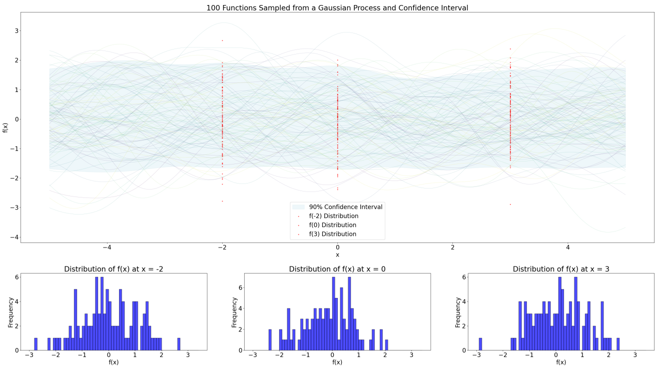


Figure 1: Gaussian Process, $m = 0$, $k = \text{RBF}$, 100 Functions sampled

Gaussian Process

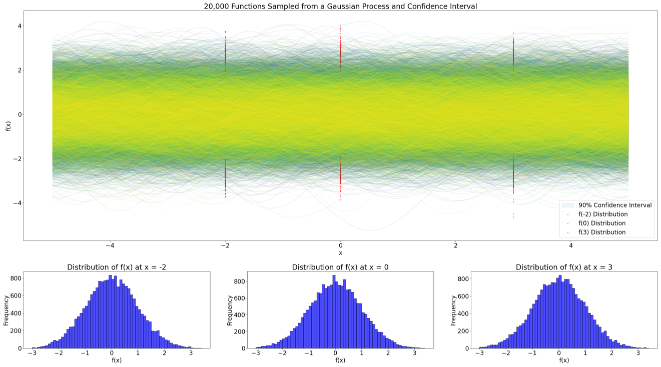
There are restriction to the functions we can sample: at each point $x_i \in \mathcal{X}$, the function

$$f(x_i) \sim \mathcal{N}(\mu(x_i), \sigma(x_i)^2),\ \forall f \in \mathcal{GP}(m(\cdot), k(\cdot,\cdot))$$

Gaussian Process

There are restriction to the functions we can sample: at each point $x_i \in \mathcal{X}$, the function

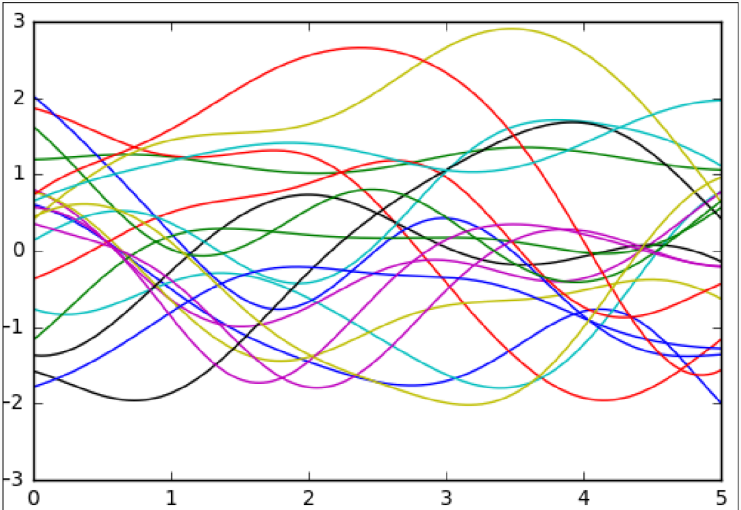
$$f(x_i) \sim \mathcal{N}(\mu(x_i), \sigma(x_i)^2),\ \forall f \in \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$$



GP Kernels: gives you just about anything!

The RBF Kernel:

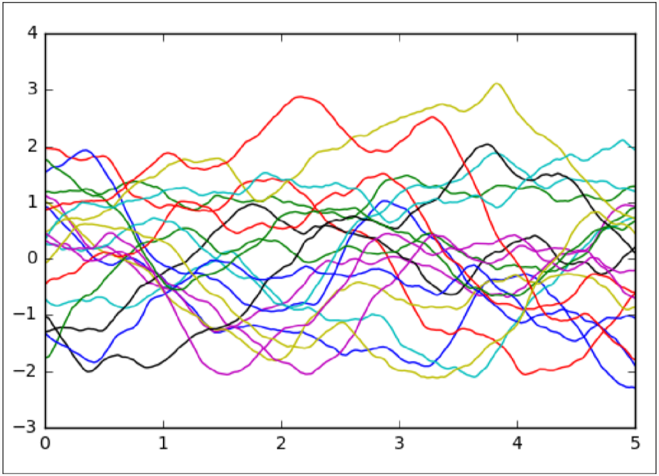
$$k(x, k') = C \exp -\frac{1}{2} K^2 (x - x')^2$$



GP Kernels: gives you just about anything!

The Matern $\frac{3}{2}$ kernel:

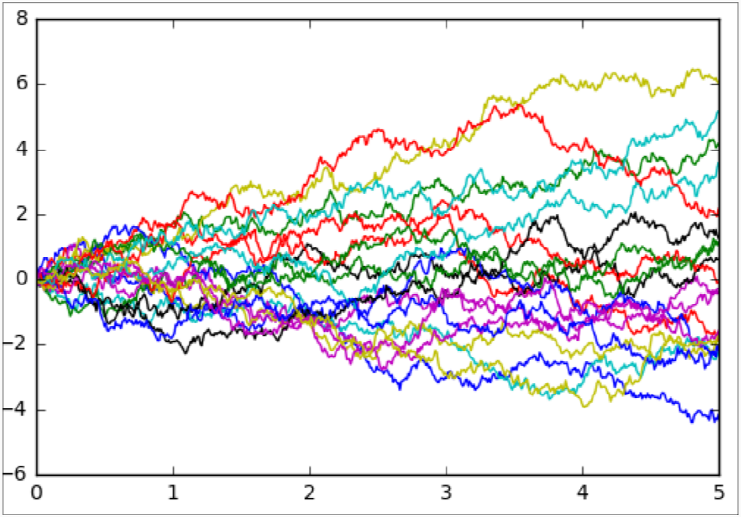
$$k(x, k') \sim (1 + |x - x'|) \exp -|x - x'|$$



GP Kernels: gives you just about anything!

The Brownian kernel:

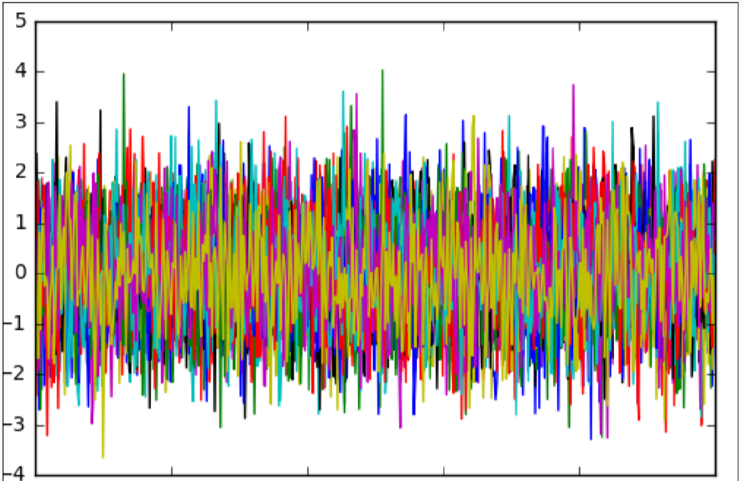
$$k(x, x') = \min\{x, x'\}$$



GP Kernels: gives you just about anything!

White noise

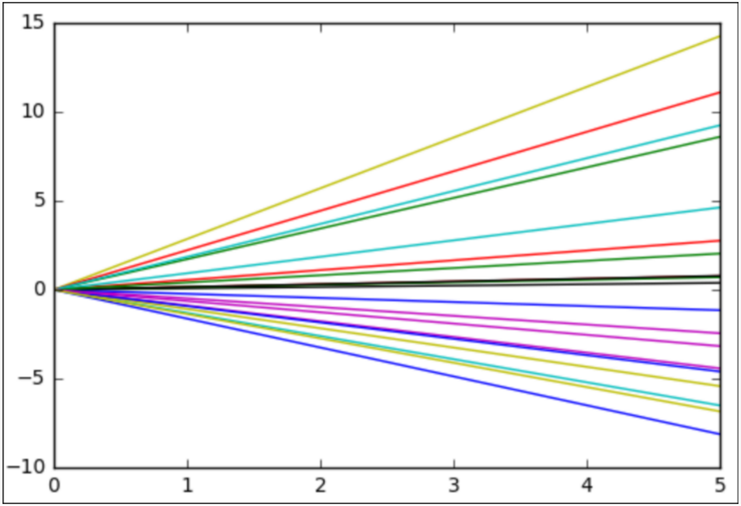
$$k(x, x') = \begin{cases} 1 & \text{if } x = x' \\ 0 & \text{otherwise} \end{cases}$$



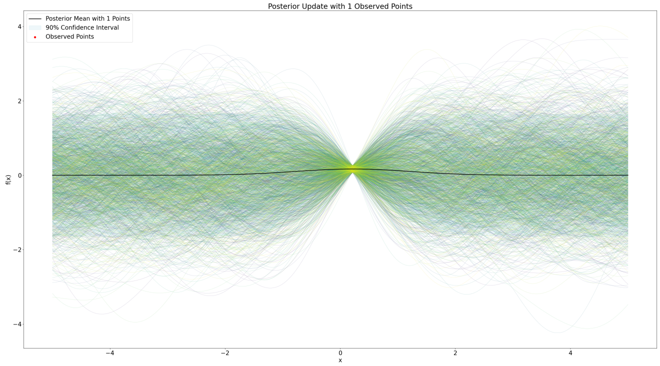
GP Kernels: gives you just about anything!

Linear kernel:

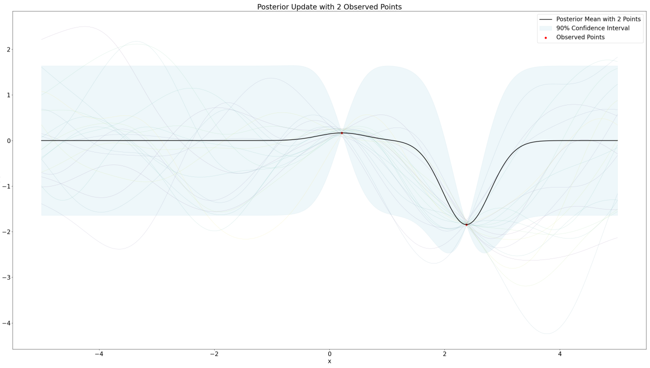
$$k(x, x') = xx'$$



Bayesian Optimization



Bayesian Optimization



Bayesian Optimisation

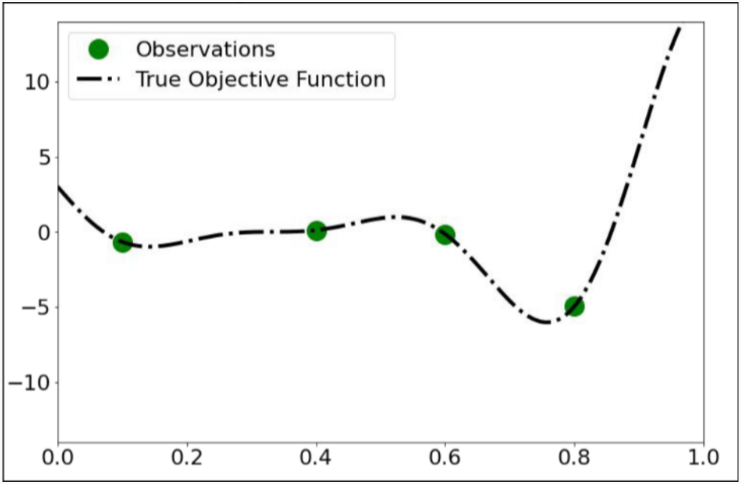
The goal is to find a point that fits our requirement in as little experiments as possible...

Bayesian Optimisation

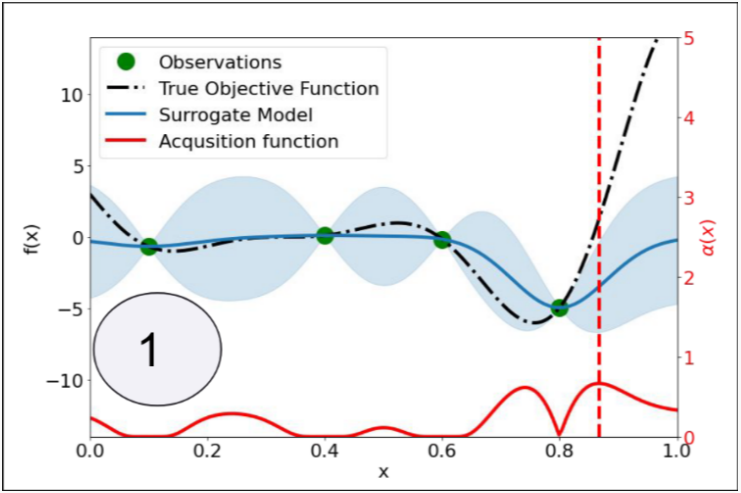
The goal is to find a point that fits our requirement in as little experiments as possible...

And be confident that we have found the optimal solution when we stop searching...

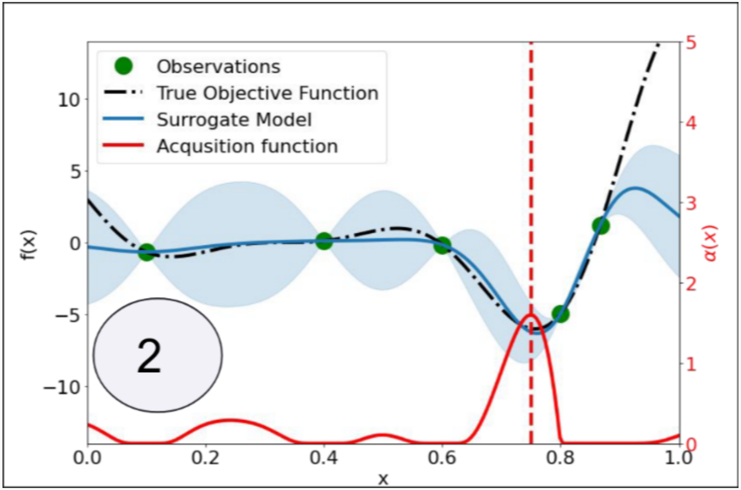
Bayesian Optimisation Example 1/5



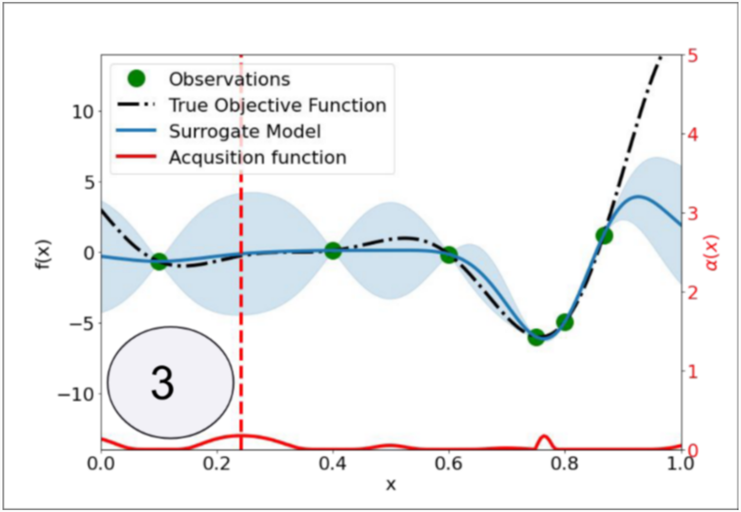
Bayesian Optimisation Example 2/5



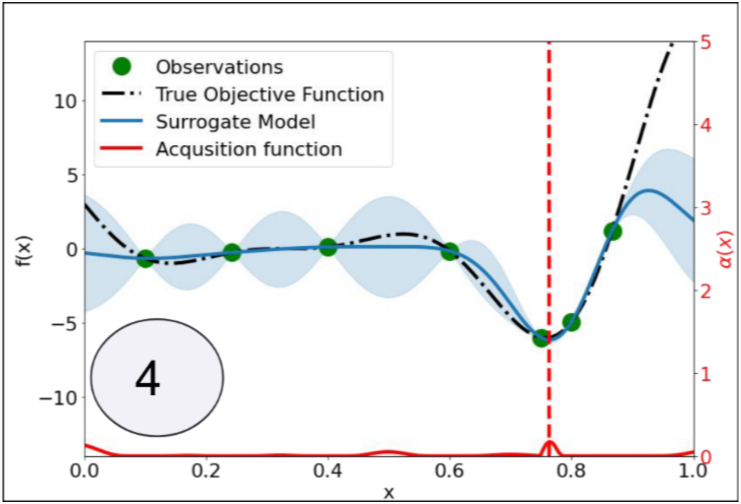
Bayesian Optimisation Example 3/5



Bayesian Optimisation Example 4/5



Bayesian Optimisation Example 5/5



Limitations of BO

- Convergence is hard when input dimension $> 10/20/30$
- How do we tweak the model if we *actual know something about the system*?

Structured Bayesian Optimisation

Structured BO

- Express domain knowledge as probability model
- Use standard BO techniques to sample unknown parameters

Structured BO

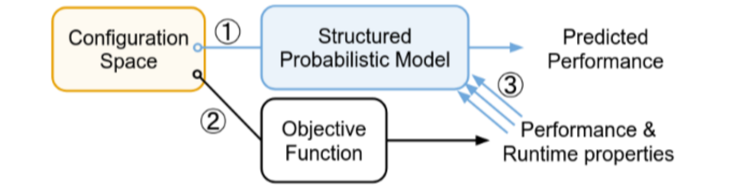

Configuration Space
①
Structured Probabilistic Model
Predicted Performance
②
Objective Function
③
Performance & Runtime properties

Figure 3.1: Procedure of structured Bayesian optimisation.

Structured BO: Semi–Parametric Model

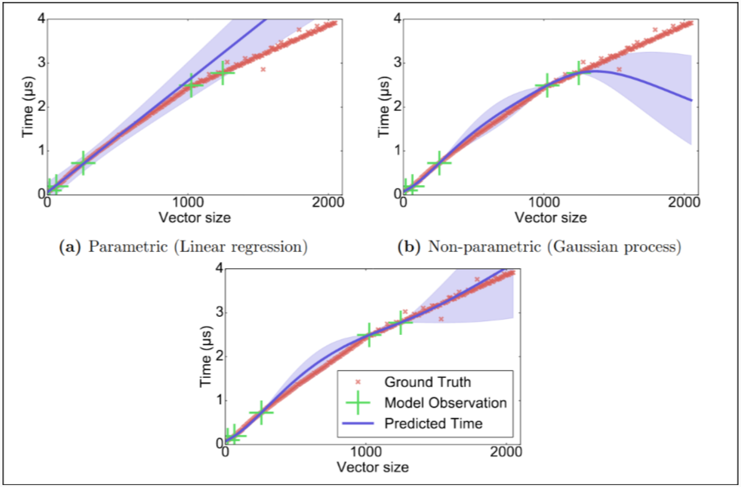


Figure 3: Prediction of runtime for inserting number into sorted vector.

Case Study: Garbage Collector Latency

- Cares about 99% Percentile latency
- Input parameters: young generation flag `ygs`, survivor ratio `sr` and max tenuring threshold `mtt`.
- Observation: rate of garbage collection is roughly inversely proportional to eden size.
- Use Gaussian Process to model the difference.

Case Study: GC Latency (cont.d)

```
class GCRateModel{
  GCRateModel(){
    // Prior distribution on the parameters
    allocated_mbs_per_sec = uniform_draw(0.0, 5000.0);
    gp.stdev(uniform_draw(3.0,30.0));
    gp.linear_scales({uniform_draw(0.0, 15000.0),
                      uniform_draw(0.0, 20.0)});
    gp.noise(uniform_draw(0.001, 0.01));
  }

  double parametric(int ygs, int sr){
    //Compute the size of eden used by the JVM
    double eden_size = ygs * sr / (sr + 2);
    return allocated_mbs_per_sec / eden_size;
  }

  double predict(int ygs, int sr, int mtt) {
    return gp.predict({ygs, sr, mtt}) +  parametric(ygs, sr);
  }

  double observe(int ygs, int sr, int mtt, double observed_rate){
    return gp.observe({ygs, sr, mtt},
                      observed_rate - parametric(ygs,  sr));
  }

  double allocated_mbs_per_sec;
  GaussianProcess gp;
}
```

Case Study: GC Latency (cont.d)

Task is split into compartmentalized subtasks, each subtask can be individually measured and evaluated.

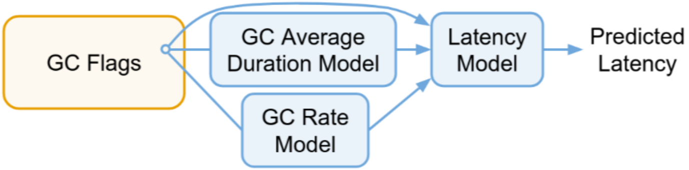
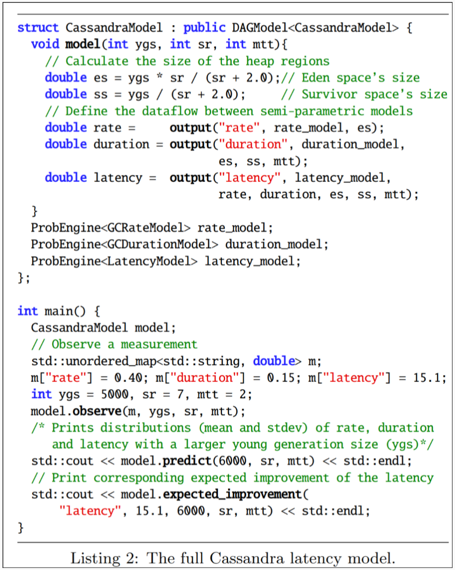


Figure 2: Dataflow of our garbage collection model

What's (poentitally)nice about it: in larger tasks, we can split up the input parameter space*.

Case Study: GC Latency (cont.d)



```
struct CassandraModel : public DAGModel<CassandraModel> {
  void model(int ygs, int sr, int mtt){
    // Calculate the size of the heap regions
    double es = ygs * sr / (sr + 2.0);// Eden space's size
    double ss = ygs / (sr + 2.0);     // Survivor space's size
    // Define the dataflow between semi-parametric models
    double rate =     output("rate", rate_model, es);
    double duration = output("duration", duration_model,
                             es, ss, mtt);
    double latency =  output("latency", latency_model,
                             rate, duration, es, ss, mtt);
  }
  ProbEngine<GCRateModel> rate_model;
  ProbEngine<GCDurationModel> duration_model;
  ProbEngine<LatencyModel> latency_model;
};

int main() {
  CassandraModel model;
  // Observe a measurement
  std::unordered_map<std::string, double> m;
  m["rate"] = 0.40; m["duration"] = 0.15; m["latency"] = 15.1;
  int ygs = 5000, sr = 7, mtt = 2;
  model.observe(m, ygs, sr, mtt);
  /* Prints distributions (mean and stdev) of rate, duration
     and latency with a larger young generation size (ygs)*/
  std::cout << model.predict(6000, sr, mtt) << std::endl;
  // Print corresponding expected improvement of the latency
  std::cout << model.expected_improvement(
      "latency", 15.1, 6000, sr, mtt) << std::endl;
}
```

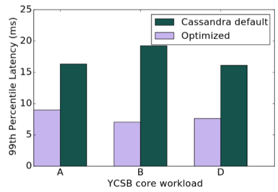
Listing 2: The full Cassandra latency model.

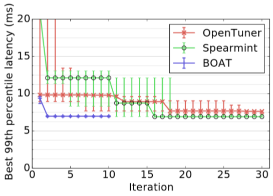
Results & Evaluation

Test 1: Garbage Collection

- Tunable Parameters:
  - Young generation flag `ygs`
  - Survivor ratio `sr`
  - Max tenuring threshold `mtt`.
- Probability Model (Intermediate Results)
  - GC Rate
  - GC Duration

Test 1: Garbage Collection Results





Test 2: Distributed ML Training

- Input parameters:
  - Machine configurations
  - NN Architecture
  - Batch Size
- Tunable Parameters:
  - Subset of machines to use as workers
  - Subset of machine to use as parameter server
  - Partition of workload between machines
- Proability Model (Intermediate Results)
  - Individual device compute time
  - Individual machine compute time
  - Communication Cost

Test 2: Distributed ML Training (Cont.d)

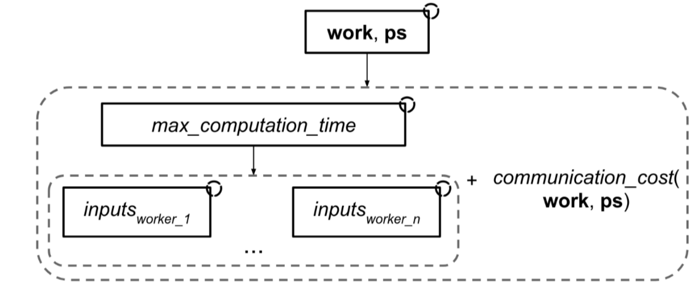


Figure 7.5: Top decompositions of the neural network case study.

Test 2: Distributed ML Training (Cont.d)

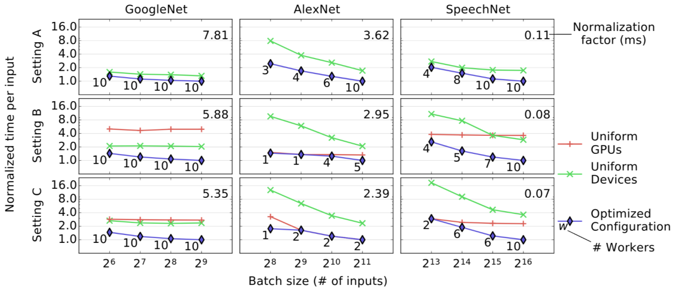
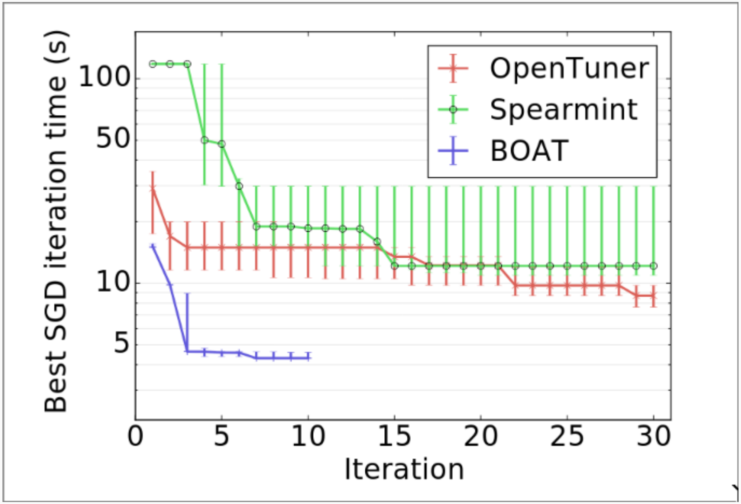


Figure 7: Normalized time per input (lower is better) of simple and optimized configurations on each experiment. Within each sub-graph, results are normalized by the best achieved time per input. This is always the one of the optimized configuration on the largest batch size (the lower right point of each sub-graph). The normalization factor, i.e. the best time per input, is shown at the top right of each sub-graph in milliseconds. For each optimized configuration, we report the number of workers used.

Test 2: Distributed ML Training (Cont.d)



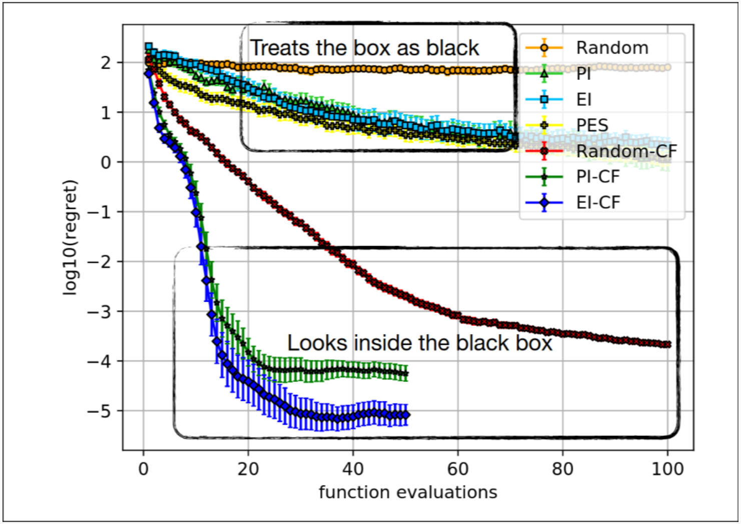
Conclusion

Conclusion

- (+) Interesting way to address parameter size limit of BO
- (+) Interesting way to inject domain knowledge into BO setting
- (+) Good arguments made on how this method addresses the dimensionality challenge faced by BO
- (-) Paper is not self–complete: semi–parametric section, DAG section poorly described in paper; more information in thesis
- (-) Did not discuss the extra compute cost (if any) of this method per iteration compared to standard BO

Related Works

Grey Box Optimisations



Grey Box Optimisations

- Astudillo & F., ”Bayesian Optimization of Composite Functions”, ICML 2019
- Wu, Toscano-Palmerin, Wilson, F., “Practical Multi-fidelity Bayesian Optimization of Iterative Machine Learning Algorithms” UAI 2019
- Toscano-Palmerin, F. ”Bayesian Optimization with Expensive Integrands”, in submission, arxiv 1803.08661
- Wu, Poloczek, Wilson, Frazier, ”Bayesian Optimization with Gradients” NIPS 2017
- Poloczek, Wang, F., “Multi-Information Source Optimization” Neural Information Processing Systems NIPS 2017