BOAT: Building Auto-Tuners with Structured Bayesian Optimization

Valentin Dalibard, Michael Schaarschmidt, Eiko Yoneki

Presented by Harrison Brown for R244

Auto-tuners

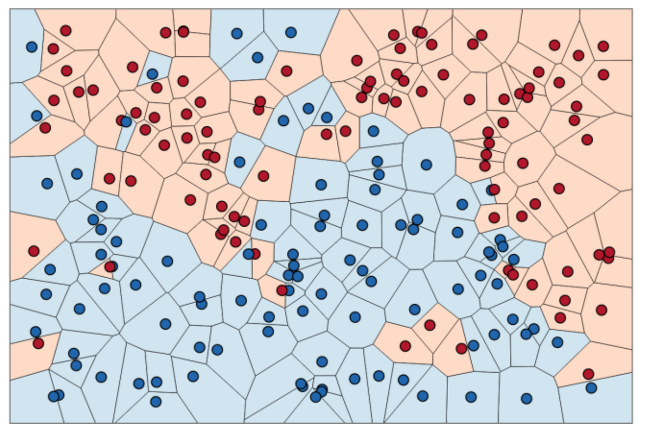
- Difficult to manually tune complex configuration parameters for various problems
  - Compiler flags, configuration files, number and assignment of machines, etc
- Can expose configuration parameters and performance metrics to black box optimizers
  - May take thousands of iterations on complex problems
    - For systems problems with long evaluation times this process fails
  - Does not leverage any contextual information about problem
- OpenTuner – ensembles of various algorithms (evolutionary, hill climbing, etc)
- Spearmint – traditional Bayesian Optimization

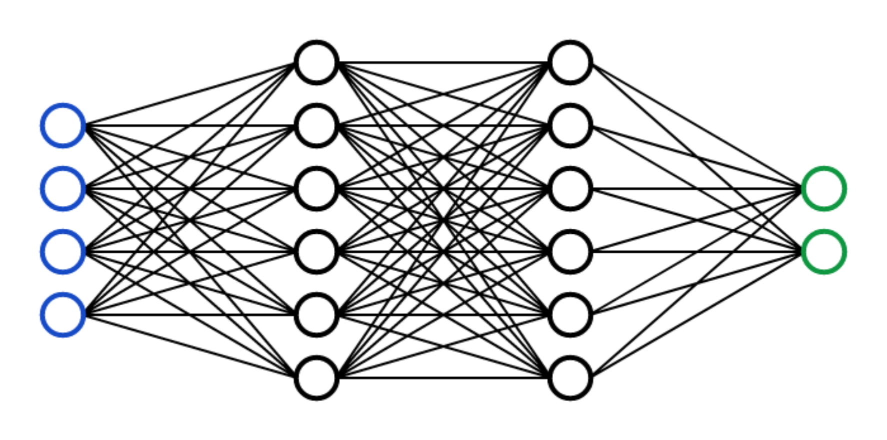
Key Terms

- Gaussian Process – collection of random variables
  - Every finite linear combination of variables is normally distributed
- Parametric models – fixed number of parameters
  - Feedforward neural networks, linear regression, logistic regression
- Non-parametic models – unbounded number of parameters
  - K-nearest neighbors

Algorithm 1 The Bayesian optimization methodology

Input: Objective function $f()$
Input: Acquisition function $\alpha()$
Initialize the Gaussian process G
for $i = 1, 2, \ldots$ **do**
    Sample point: $\mathbf{x}_t \leftarrow \arg\max_{\mathbf{x}} \alpha(G(\mathbf{x}))$
    Evaluate new point: $y_t \leftarrow f(\mathbf{x}_t)$
    Update the Gaussian process: $G \leftarrow G \mid (\mathbf{x}_t, y_t)$
end for





BOAT Contributions

- Novel algorithm – Structured Bayesian Optimization
  - Structured probabilistic model provided by developer
    - Discards regions of low performance where traditional Bayesian Optimization over explores
    - Semi-parametric model – developer provides parametric parts to describe general behavior
- BOAT – a framework to allow developers to build auto-tuners for their systems
  - To be used in situations where generic autotuners fail
  - Model allows for probabilistic inference
    - Can make predictions without large computational cost

Using BOAT and SBO

- Configuration Space
- Objective function and runtime measurements
- Probalistic model of system behavior
    - Semi-parametric model
        - Constructor – prior distribution to sample model parameters
        - A parametric function for a given input that returns a prediction
    - DAG model, allows combination of multiple semi-parametric models
        - Exploits conditional independence to train independently given the measured outputs
        - Allows maximization of expected improvement in SBO

```
struct GCRateModel : public SemiParametricModel<GCRateModel> {
  GCRateModel() {
    allocated_mbs_per_sec =
     std::uniform_real_distribution<>(0.0, 5000.0)(generator);
    // Omitted: also sample the GP parameters
  }
  double parametric(double eden_size) const {
    // Model the rate as inversly proportional to Eden's size
    return allocated_mbs_per_sec / eden_size;
  }
  double allocated_mbs_per_sec;
};

int main() {
  // Example: observe two measurements and make a prediction
  ProbEngine<GCRateModel> eng;
  eng.observe(0.40, 1024);  // Eden: 1024MB, GC rate: 0.40/sec
  eng.observe(0.25, 2048);  // Eden: 2048MB, GC rate: 0.25/sec
  // Print average prediction for Eden: 1536MB
  std::cout << eng.predict(1536) << std::endl;
}
```

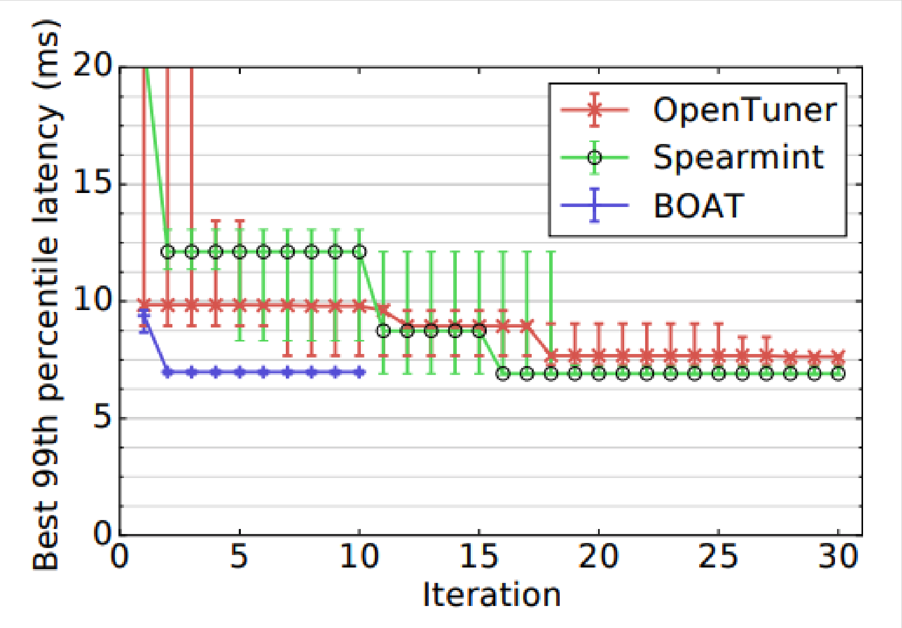
```
struct CassandraModel : public DAGModel<CassandraModel> {
  void model(int ygs, int sr, int mtt){
    // Calculate the size of the heap regions
    double es = ygs * sr / (sr + 2.0);// Eden space's size
    double ss = ygs / (sr + 2.0);     // Survivor space's size
    // Define the dataflow between 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;
};
```

BOAT Recommended Usage

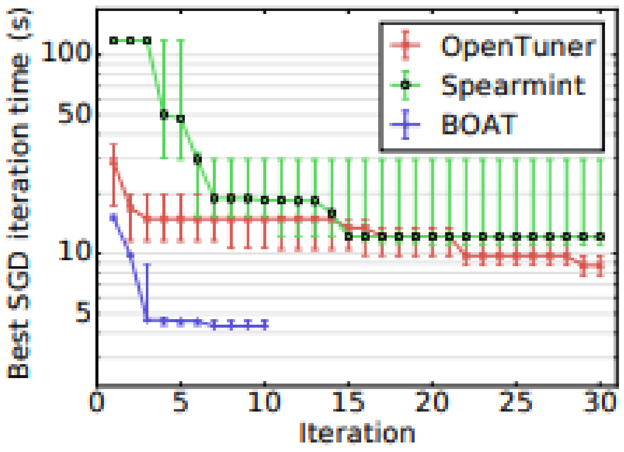
- Initially use generic probabilistic model – regular Bayesian optimization
- Incrementally add structure until convergence
  - Unclear on how long this process typically takes

Java Garbage Collection Case Study

- Tuning garbage collection flags of JVM database (Cassandra)
  - Only 3 parameters, very small domain
- Objective: 99th Percentile Latency using YCSB cloud benchmark
- Spearmint Converges within 16 iterations (4 hours)
- BOAT converges to within 10% of best-found performance by 2nd iteration



Neural Network Training Case Study



- In Tensorflow, users must set what available machines to be used and assign work
- Input: NN architecture, available machines, batch size
- Tuning synchronous distributed SGD
  - Parameters: worker machines, parameter servers, workload partition
  - Objective: minimize average iteration time
- OpenTuner only marginally better than uniform GPUs assignment (9.82s)
- BOAT completed within 2 hours, significant gains if architectures take weeks to train

BOAT Impact

Novel algorithm and framework for probalistic models

- Easy to build probalistic models with little effort

Significant gains can be made on complex problems such as neural network tuning

- Useful as black box optimizers may often fail in these domains
- If a developer has contextual knowledge, that should be leveraged

BOAT Criticisms

BOAT does not give information about performance with incorrect contextual information

Niche contribution – enough knowledge to provide model, not enough to set the configuration parameters

Motivation states "auto-tuners like [...] OpenTuner [...] usually require thousands of evaluations"

- More evidence is warranted in form of case studies / experiments
  - OpenTuner used 7 projects
  - Time versus iterations. BO has high iteration overhead

OpenTuner and Spearmint - Python, C++ user friendly

Performance gains vs usability

References

[1] V. Dalibard, M. Schaarschmidt, and E. Yoneki: BOAT: Building Auto-Tuners with Structured Bayesian Optimization, WWW, 2017.

[2] Jasper Snoek, Hugo Larochelle, and Ryan Prescott Adams. Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012.

[3] Jason Ansel et al. Opentuner: an extensible framework for program autotuning. In *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, pages 303–316. ACM, 2014.