Accelerating hyperparameter optimization using performance prediction on a heterogeneous HPC system

July 14th, 2023
Juan P. García Amboage, Eric Wulff, Maria Girone
CERN
Outline

- Basic notions of Hyperparameter Optimization
  - Deep Learning
  - What is Hyperparameter Optimization?
  - Key ideas and challenges
  - HPO steps
  - Hyperband algorithm

- Performance Prediction
  - Key ideas
  - Integration with HPO

- Quantum Annealing & Q-SVR

- Our work
  - Context
  - Performance prediction for MLPF
  - Swift-Hyperband
  - Algorithm comparison
  - Distributed Swift-Hyperband

- Conclusions and further work
Basic notions of Hyperparameter Optimization
Deep Learning

Perceptron

\[ g(\sum x_i \cdot w_i) \]

Output

Bias

Weights

Activation function: \( g \)

Numerical input

Hidden layers

Numerical output

Deep Neural Network

Training: minimize loss function → measures the difference between the real and the predicted outcome

0.87 prob. of being a “3”

“Blah...”

“do my homework”
What is Hyperparameter Optimization?

**Hyperparameters** (HPs): parameters of a model that can not be trained.

**Hyperparameter Optimization** (HPO): process of automatically searching for optimal hyperparameter configurations for a given model.
Key ideas and challenges

Given a model $M$, its loss function $f_M(w, \theta)$ is dependent on:

- $w$: the model weights $\rightarrow$ training the model $\Rightarrow$ search $\arg\min_w f_M(w, \theta)$.
- $\theta$: the model HPs $\rightarrow$ HPO $\Rightarrow$ search $\arg\min_{\theta} f_M(w, \theta)$.

- “Black box” optimization problem
- No straightforward update rule for $\theta$
- Many evaluations of $f_M$ needed

$f_M(w, \theta)$ is not differentiable w.r.t $\theta$
Key ideas and challenges

- Evaluating $f_M$ for a given configuration $\theta$ is **computationally expensive** as it requires to fully train the model $M$ under the configuration $\theta$.

- **Choosing the HPs by hand**:  
  - Requires expert knowledge and is time consuming $\times$.  
  - Usually leads to suboptimal performance of the model $\times$.

- **Automated HPO process**:  
  - Can significantly increase the performance of the model $\checkmark$  
  - Requires to evaluate many configurations  
  - Compute-resource intensive $\Rightarrow$ calls for **HPC resources**.
HPO Steps

Define HPs search space
- select HPs to optimize and their ranges
  - human responsibility

Select trials from the search space
- Random search
- Grid search
- Bayesian opt.
  - automated

Evaluate the trials
- Expensive!!
  - usually not all trials are fully evaluated
  - automated

Select best trial
  - automated

step to be accelerated using performance prediction
HPO Steps

Define HPs search space
- select HPs to optimize and their ranges
  - human responsibility

Select trials from the search space
- Random search
- Grid search
- Bayesian opt.
  - automated

Evaluate the trials
- Expensive!!
- usually not all trials are fully evaluated
  - automated

Select best trial
  - automated

step to be accelerated using performance prediction
Hyperband\textsuperscript{1} is an “early stopping” evaluation algorithm that consists in:

1. **Partially train** some trials up to a decision point.

2. Evaluate their performance and **throw out** the worst x%.

3. **Repeat** 1\(\rightarrow\)2 **until** only one configuration remains or the target epoch is reached.

4. **Repeat** 1\(\rightarrow\)3 for different sets of decision points.

\[1\] Li et al.
Key ideas

Predict the final performance of a model given its partial learning curve.

➢ **Performance predictor:** “meta-model” is used to predict the final loss of the target model.

➢ The performance predictor can be a simple regressor (Baker et al.):
  - Support Vector Regression ← the one we use
Key ideas

- Using **performance prediction** can **accelerate** the evaluation step in **HPO**.
  - The meta-model provides a “cheap” evaluation of $f_M(w, \theta)$.

- The performance predictor also has to be trained
  - Must be **fast to train**.
  - The **training samples** come from **fully trained trials**.

**Saved 75 epochs** of the target model!
Integration with HPO

Simple approach

Generate $N$ random configurations

- Fully train $M<<N$ trials
- Partially train $M-N$ trials

Train SVR

Predict performance

- Fully train only the most promising trials

- Easily parallelizable

- Benefit from HPC resources:
  - GPU nodes for the model training
  - CPU for SVR training, predicting...

Select the best fully trained trial
Integration with HPO

Fast-Hyperband Algorithm (Baker et al.)

➢ Hyperband + performance prediction.

➢ Adds a decision point at every epoch of each Hyperband round.

➢ Uses the predictions to estimate each trials probability of being promoted to the next round.

➢ Discards trials before finishing the round if their probability is low.
Integration with HPO

Fast-Hyperband Algorithm (Baker et al.)

➢ Hyperband + performance prediction.

➢ Adds a decision point at every epoch of each Hyperband round.

➢ Uses the predictions to estimate each trials probability of being promoted to the next round.

➢ Discards trials before finishing the round if their probability is low.
Quantum Annealing & Q-SVR
**Quantum Annealing**

**Quantum Annealer**: quantum computer that only solves QUBO problems.

**QUBO formulation:**

Minimize: $f(x) = \sum_{i<j}^{N} Q_{i,j} x_i x_j + \sum_{i}^{N} Q_{i,i} x_i$

$x_i \in \{0,1\}$ and $Q$ is a $N \times N$ upper triangular matrix.
Quantum SVR

➢ **Q-SVR**: re-formulation of SVR model that can be trained in a Quantum Annealer. ([Pasetto et al.](#))

➢ **In theory**: Q-SVR training is $O(N)$ and SVR is $O(N^3)$, $N=$#training samples. ([Date et al.](#))

➢ **In practice**:
  ○ Currently no time advantage from Q-SVR.
  ○ **Limited training size**: $\sim$20 samples.
Our work
➢ Explore the applicability of performance prediction for MLPF using classical HPC and Quantum resources.

➢ **MLPF**: neural network for particle flow reconstruction.

➢ **CoE RAISE**: The European Center of Excellence in Exascale Computing "Research on AI- and Simulation-Based Engineering at Exascale".

➢ **JSC D-WAVE Quantum Annealer**: Jülich Supercomputing Centre provides access to a Quantum Annealer for the RAISE project.
Performance prediction of MLPF

➢ **Learning curve dataset** of 297 MLPF random configurations trained for 100 epochs.
   - Trained on Delphes dataset.
   - $HP_1, HP_2, \ldots, HP_7$, $Loss_0, Loss_1, \ldots, Loss_{99}$

➢ Use $R^2$ as evaluation metric for SVR and Q-SVR.

**Hyperparameter search space for MLPF**

<table>
<thead>
<tr>
<th>type</th>
<th>learning rate</th>
<th>num graph layers id</th>
<th>num graph layers reg</th>
<th>dropout</th>
<th>bin size</th>
<th>output dim</th>
<th>weight decay</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>log uniform</td>
<td>quantized uniform</td>
<td>quantized uniform</td>
<td>uniform</td>
<td>choice</td>
<td>choice</td>
<td>log uniform</td>
</tr>
<tr>
<td></td>
<td>(1e-6, 3e-2)</td>
<td>[0, 4]</td>
<td>[0, 4]</td>
<td>(0.0, 0.5)</td>
<td>[8, 16, 32, 64, 128]</td>
<td>[8, 16, 32, 64, 128, 256]</td>
<td>(1e-6, 1e-1)</td>
</tr>
</tbody>
</table>

Some MLPF Learning Curves
Performance prediction of MLPF

➢ VERY PROMISING RESULTS for Q-SVR and SVR.

MLPF performance predictor $R^2$ vs known fraction of learning curve

MLPF predicted loss vs MLPF true loss

Best Q-SVR true vs predicted test values

train size = 20, known fraction of $I_c = 0.25$

$R^2 = 0.948$
Swift-Hyperband

- **Fast-Hyperband**: not suitable for integration with Q-SVRs.

- **Swift-Hyperband**: new approach to combine performance prediction with Hyperband.

<table>
<thead>
<tr>
<th>Fast-Hyperband</th>
<th>Swift-Hyperband</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple decision points inside each round</td>
<td>Only 1 decision point inside each round</td>
</tr>
<tr>
<td>Estimates $\sigma$ for every SVR</td>
<td>No need to estimate $\sigma$</td>
</tr>
<tr>
<td>Sequential</td>
<td>Easily parallelizable</td>
</tr>
<tr>
<td>Trains <strong>many</strong> SVRs</td>
<td>Trains <strong>few</strong> SVRs</td>
</tr>
<tr>
<td>Not suitable for Q-SVRs</td>
<td>Suitable for Q-SVRs</td>
</tr>
</tbody>
</table>
Swift-Hyperband

➢ One extra decision point inside each round
➢ At the beginning of the round some trials are fully trained to define a threshold.
➢ The other trials are partially trained.
➢ If their predicted loss is lower than the threshold the trials are stopped before completing the round.

Full and partial trainings can be done in parallel.
Algorithm Comparison

➢ Simulated results using learning curve datasets

### MLPF for Delphes - 7 HPs

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean Loss</th>
<th>Mean Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperband</td>
<td>442.42</td>
<td></td>
</tr>
<tr>
<td>Fast-Hyperband</td>
<td>442.42</td>
<td></td>
</tr>
<tr>
<td>Swift-Hyperband SVR</td>
<td>442.42</td>
<td></td>
</tr>
<tr>
<td>Swift-Hyperband Q−SVR</td>
<td>442.42</td>
<td></td>
</tr>
</tbody>
</table>

### LSTM for PTB - 2 HPs

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Mean Perplexity</th>
<th>Mean Epochs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperband</td>
<td>66.03</td>
<td></td>
</tr>
<tr>
<td>Fast-Hyperband</td>
<td>66.11</td>
<td></td>
</tr>
<tr>
<td>Swift-Hyperband SVR</td>
<td>66.10</td>
<td></td>
</tr>
<tr>
<td>Swift-Hyperband Q−SVR</td>
<td>65.71</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm Comparison

Simulated results using learning curve datasets

CNN for CIFAR-10 - 5 HPs

- Hyperband: 0.8169
- Fast-Hyperband: 0.8166
- Swift-Hyperband: 0.8175
- Swift-Hyperband SVR: 0.8176

CNN for SVHN - 9 HPs

- Hyperband: 0.935
- Fast-Hyperband: 0.933
- Swift-Hyperband SVR: 0.934
- Swift-Hyperband Q-SVR: 0.935
Distributed Swift-Hyperband

- **1 CPU node** that coordinates the workflow.
- **Multiple GPU worker nodes** for training the trials.
- **Quantum Annealer** that trains the performance predictor.
- MPI + dwave-ocean-sdk.

Full and partial trainings inside each round can be done in parallel.

![Graph showing speedup vs. number of GPU workers]
Distributed Swift-Hyperband

- **1 CPU node** that coordinates the workflow.
- **Multiple GPU worker nodes** for training the trials.
- **Quantum Annealer** that trains the performance predictor.
- MPI + dwave-ocean-sdk.

Full and partial trainings inside each round can be done in **parallel**.
Distributed Swift-Hyperband

- **1 CPU node** that coordinates the workflow.
- **Multiple GPU worker nodes** for training the trials.
- **Quantum Annealer** that trains the performance predictor.
- MPI + dwave-ocean-sdk.

Full and partial trainings inside each round can be done in **parallel**.
Conclusions and further work
Conclusions

➢ Swift-Hyperband is a new algorithm that integrates performance prediction with HPO suitable for HPC environments.

➢ Swift-Hyperband holds good potential for accelerating HPO:
  ○ Future MLPF cycles.
  ○ Even other models.

➢ It is possible to integrate quantum resources and heterogeneous HPC and produce useful results.
Further work

➢ Test Swift-Hyperband for other target models.
  ○ Simulation.
  ○ HPC.

➢ Find ways to solve Swift-Hyperband scalability issues.
  ○ Integrate performance predictors with other HPO algorithms.
  ○ Dynamic resource allocation.

➢ Study deeper:
  ○ The Q-SVR model.
  ○ Why Swift-Hyperband can find better configurations than Hyperband.
The CoE RAISE project has received funding from the European Union’s Horizon 2020 – Research and Innovation Framework Programme H2020-INFRAEDI-2019-1 under grant agreement no. 951733
From SVR to Q-SVR formulation \cite{Pasetto2011}

### Classical SVR primal formulation

$$\text{minimize: } \frac{1}{2}||w||^2 + C \sum_{i=0}^{N-1} (\xi_i + \xi_i^*)$$

\[ y_i - w^T x_i - b \leq \epsilon + \xi_i^* \quad \forall i \in \{0, \ldots, N-1\} \]

\[ w^T x_i + b - y_i \leq \epsilon + \xi_i \quad \forall i \in \{0, \ldots, N-1\} \]

\[ \xi_i, \xi_i^* \geq 0 \quad \forall i \in \{0, \ldots, N-1\} \]

**predictions:** \( y = w^T x + b \)

### Classical SVR dual formulation

$$\text{minimize: } \frac{1}{2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} (\alpha_n - \hat{\alpha}_n)(\alpha_m - \hat{\alpha}_m)k(x_n, x_m) - \epsilon \sum_{n=0}^{N-1} (\alpha_n + \hat{\alpha}_n) + \sum_{n=0}^{N-1} (\alpha_n - \hat{\alpha})y_n$$

\[ \sum_{n=0}^{N-1} (\alpha_n - \hat{\alpha}_n) = 0 \]

\[ 0 \leq \alpha_n, \hat{\alpha}_n \leq C \quad \forall n \in \{0, \ldots, N-1\} \]

**predictions:**

\[ y = \sum_{n=0}^{N-1} (\alpha_n - \hat{\alpha}_n)k(x_n, x_m) + b \]

**calculate b:**

\[ b = y_n - \epsilon - \sum_{m=1}^{N} (\alpha_m - \hat{\alpha}_m)k(x_n, x_m) \]

### QUBO formulation

**1.** Add restriction as penalty term

$$\text{minimize: } \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} Q_{ij} a_i a_j$$

\[ \xi \left( \sum_{n=0}^{N-1} (\alpha_n - \hat{\alpha}_n) \right)^2 \]

“Ignore” the 2\(^{nd}\) restriction

\[ \sum_{k=0}^{K-1} B^{k-k_0} a_{K+n+k} \]

**2.** Encode SVR variables using binary variables

\[ \alpha_n = \sum_{k=0}^{K-1} B^{k-k_0} a_{K+n+k}, \quad \hat{\alpha}_n = \sum_{k=0}^{K-1} B^{k-k_0} a_{K(n+n)+k} \]

**3.** Resulting problem with binary variables and without restrictions

**4.** QUBO matrix for the canonical formulation:

\[ Q_{ij} = \begin{cases} \hat{Q}_{ij} & \text{if } i < j \\ \hat{Q}_{ij} & \text{if } i = j \\ 0 & \text{if } i > j \end{cases} \]