Supplemental Material

Artificial Intelligence for the Earth Systems
Two-Step Hyperparameter Optimization Method: Accelerating Hyperparameter Search by Using a Fraction of a Training Dataset
https://doi.org/10.1175/AIES-D-23-0013.1

© Copyright 2023 American Meteorological Society (AMS)
For permission to reuse any portion of this work, please contact permissions@ametsoc.org. Any use of material in this work that is determined to be “fair use” under Section 107 of the U.S. Copyright Act (17 USC §107) or that satisfies the conditions specified in Section 108 of the U.S. Copyright Act (17 USC §108) does not require AMS’s permission. Republication, systematic reproduction, posting in electronic form, such as on a website or in a searchable database, or other uses of this material, except as exempted by the above statement, requires written permission or a license from AMS. All AMS journals and monograph publications are registered with the Copyright Clearance Center (https://www.copyright.com). Additional details are provided in the AMS Copyright Policy statement, available on the AMS website (https://www.ametsoc.org/PUBSCopyrightPolicy).
Supplemental Material

For

Two-step hyperparameter optimization method: Accelerating hyperparameter search by using a fraction of a training dataset

Sungduk Yu, Mike Pritchard, Po-Lun Ma, Balwinder Singh, Sam Silva

a UCI First affiliation, City, State/Territory, Country (if outside U.S.)

b NVIDIA Second affiliation, City, State/Territory, Country (if outside U.S.)

c PNNL Third affiliation, City, State/Territory, Country (if outside U.S.)

d University of Southern California, Department of Earth Sciences, Los Angeles, CA

Corresponding author: Sungduk Yu, sungduk@uci.edu

Contents of this file:

   Text S1
   Tables S1
   Figures S1 to S11
Text S1: Code example to set up a distributed mode in Keras Tuner, in a SLURM-managed high-performance computing cluster (HPC).

Keras Tuner provides a high-level interface that enables distributed search mode with only four environmental variables. These include three Keras Tuner-specific variables (1-3) and one CUDA-related variable (4):

1) KERASTUNER_TUNER_ID: A unique ID assigned to manager and worker processes, with "chief" used for the manager process.
2) KERASTUNER_ORACLE_IP: IP address or hostname of the manager (chief) process.
3) KERASTUNER_ORACLE_PORT: Port number of the manager (chief) process.
4) CUDA_VISIBLE_DEVICES: Local GPU device ID to control the number and selection of GPUs assigned to a single process.

These four environmental variables are assigned dynamically depending on the availability of computing resources in an HPC. Accordingly, it can be technically challenging to write scripts that automatically set up correct values for the above four environmental variables. To address this, we provide code examples based on the scripts used in our case study.

Our case study was conducted on PSC Bridges-2 (https://www.psc.edu/resources/bridges-2/), which has 8 GPUs per node in its GPU partition. We used two nodes (16 GPUs in total) and assigned one GPU to each worker process. To automatically set the above four environmental variables, we wrote three scripts:

- Script 1. sbatch-keras-tuner.sh
  : SLURM job submission script that launches parallel jobs (in our case, ‘run-dynamic.sh’) using srun. The following options need to be set based on a user’s needs:
    o partition: name of a GPU partition
    o nodes: number GPU nodes
    o gpus: number of total GPUs processes, i.e., nodes x GPUs/node
    o ntasks: number of tasks, i.e., gpus +1. Extra one process accounts for the process for a manager.

- Script 2. run-dynamic.sh
  : Intermediary script bridging the SLURM script (‘sbatch-keras-tuner.sh’) and the python script using Keras Tuner (‘keras-tuner-dynamic.py’). It is launched in parallel with each job step having its own SLURM-generated environmental variables, e.g., SLURM_LOCALID and SLURMD_NODENAME.
- Script 3. keras-tuner-dynamic.py
  : The python script for hyperparameter tuning using Keras Tuner, with the environmental variables (1-4) set automatically based on SLURM variables:
    o num_gpus_per_node (Line 1): the number of GPUs per node.

  <Script 1. sbatch-keras-tuner.sh>

```
#!/bin/sh

## The following options should be supplied by a user
#SBATCH --job-name=
#SBATCH --output=
#SBATCH --account=
#SBATCH --time=

## The following options should also be supplied by a user.
## However, we provide an example of using 2 nodes in PSC Bridges-2
#SBATCH --partition=GPU # Name of Bridges-2's gpu partition
#SBATCH --nodes=2     # Number of nodes
#SBATCH --gpus=16     # Total number of gpu processors:
    # i.e., <Number of nodes> times <Number of GPUs per node>
#SBATCH --ntasks=17   # Total number of tasks:
    # should be total number of GPUs plus one,
    # i.e., <Number of nodes> times <Number of GPUs per node> + 1

srun --wait=0 bash run-dynamic.sh
```

  <Script 2. run-dynamic.sh>

```
#!/bin/sh

# The following three lines are optional for a log file.
# SLURM_LOCALID and SLURMD_NODENAME are environmental variables that automatically set by SLURM.
echo "--- run-dynamic.sh ---"
echo SLURM_LOCALID $SLURM_LOCALID
echo SLURMD_NODENAME $SLURMD_NODENAME

# Run the keras-tuner script.
# Standard output and errors are printed to an individual log file.
python keras-tuner-dynamic.py > log-$SLURM_JOBID-$SLURMD_NODENAME-$SLURM_LOCALID.log 2>&1
```

  <Script 3. keras-tuner-dynamic.py>

```
def set_environment(num_gpus_per_node="8"): # 'num_gpus_per_node' should be set based on a HPC spec.
    import os
    nodename = os.environ["SLURMD_NODENAME"]
    procid = os.environ["SLURM_LOCALID"]
    print(nodename)
    print(procid)
```
```python
stream = os.popen('scontrol show hostname $SLURM_NODELIST')
output = stream.read()
oracle = output.split("n")[0]
print(oracle)
if procid==num_gpus_per_node:
    os.environ["KERASTUNER_TUNER_ID"] = "chief"
    os.environ["CUDA_VISIBLE_DEVICES"] = "0"
else:
    os.environ["KERASTUNER_TUNER_ID"] = "tuner-" + str(nodename) + "." + str(procid)
    os.environ["CUDA_VISIBLE_DEVICES"] = procid

os.environ["KERASTUNER_ORACLE_IP"] = oracle + ".ib.bridges2.psc.edu" # Use full hostname
os.environ["KERASTUNER_ORACLE_PORT"] = "8000"
print("KERASTUNER_TUNER_ID: %s"%os.environ["KERASTUNER_TUNER_ID"])
print("KERASTUNER_ORACLE_IP: %s"%os.environ["KERASTUNER_ORACLE_IP"])
print("KERASTUNER_ORACLE_PORT: %s"%os.environ["KERASTUNER_ORACLE_PORT"])

def main():
    # User's tuning code goes here, e.g.,
    # - Import necessary packages including Keras Tuner
    # - Read dataset
    # - Normalize/scale if necessary
    # - Define a 'hypermodel' (a search space is specified here)
    # - Define a 'tuner' (a search algorithm is specified here)
    # - Execute hyperparameter searching

if __name__ == '__main__':
    set_environment()
    main()
<table>
<thead>
<tr>
<th>Rank (step 2)</th>
<th>Project</th>
<th>Rank (Step 1)</th>
<th>MSE (Step 2)</th>
<th># Layers</th>
<th># Params</th>
<th>Number of nodes in hidden layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.5</td>
<td>30</td>
<td>5.937E-06</td>
<td>10</td>
<td>1.777E+06</td>
<td>1024, 512, 256, 1024, 64, 256, 512, 64, 1024, 512</td>
</tr>
<tr>
<td>2</td>
<td>0.5</td>
<td>24</td>
<td>6.307E-06</td>
<td>7</td>
<td>2.660E+06</td>
<td>1024, 2048, 256, 64, 16, 8, 128</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>38</td>
<td>6.499E-06</td>
<td>13</td>
<td>9.887E+05</td>
<td>64, 32, 512, 32, 256, 1024, 128, 1024, 16, 16, 256, 512</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>1</td>
<td>6.798E-06</td>
<td>8</td>
<td>4.775E+06</td>
<td>1024, 2048, 1024, 512, 32, 8, 64, 256</td>
</tr>
<tr>
<td>5</td>
<td>0.5</td>
<td>6</td>
<td>6.887E-06</td>
<td>9</td>
<td>9.712E+05</td>
<td>512, 512, 1024, 8, 32, 8, 32, 1024, 128</td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td>2</td>
<td>7.218E-06</td>
<td>5</td>
<td>1.462E+06</td>
<td>1024, 128, 1024, 1024, 128</td>
</tr>
<tr>
<td>7</td>
<td>0.25</td>
<td>46</td>
<td>7.387E-06</td>
<td>7</td>
<td>2.656E+06</td>
<td>512, 256, 1024, 2048, 8, 128, 1024</td>
</tr>
<tr>
<td>8</td>
<td>0.05</td>
<td>49</td>
<td>7.393E-06</td>
<td>11</td>
<td>2.443E+06</td>
<td>1024, 2048, 128, 64, 16, 64, 32, 16, 256, 16, 2048</td>
</tr>
<tr>
<td>9</td>
<td>0.005</td>
<td>15</td>
<td>7.393E-06</td>
<td>10</td>
<td>2.683E+06</td>
<td>2048, 1024, 512, 32, 64, 16, 16, 8, 16, 256</td>
</tr>
<tr>
<td>10</td>
<td>0.5</td>
<td>2</td>
<td>7.406E-06</td>
<td>11</td>
<td>2.817E+06</td>
<td>2048, 1024, 8, 512, 8, 32, 8, 512, 1024, 64, 1024</td>
</tr>
<tr>
<td>11</td>
<td>0.0005</td>
<td>26</td>
<td>7.440E-06</td>
<td>12</td>
<td>9.145E+05</td>
<td>2048, 64, 1024, 64, 32, 128, 64, 128, 256, 2048, 16, 64</td>
</tr>
<tr>
<td>12</td>
<td>0.5</td>
<td>28</td>
<td>7.574E-06</td>
<td>6</td>
<td>2.139E+06</td>
<td>256, 1024, 512, 2048, 128, 256</td>
</tr>
<tr>
<td>13</td>
<td>0.05</td>
<td>44</td>
<td>7.656E-06</td>
<td>6</td>
<td>1.384E+06</td>
<td>2048, 32, 1024, 64, 2048, 512</td>
</tr>
<tr>
<td>14</td>
<td>0.005</td>
<td>12</td>
<td>7.716E-06</td>
<td>10</td>
<td>8.498E+05</td>
<td>1024, 512, 512, 16, 512, 16, 512, 16, 32, 64</td>
</tr>
<tr>
<td>15</td>
<td>0.25</td>
<td>20</td>
<td>7.760E-06</td>
<td>6</td>
<td>1.183E+05</td>
<td>256, 256, 16, 32, 16, 2048</td>
</tr>
<tr>
<td>16</td>
<td>0.5</td>
<td>49</td>
<td>8.025E-06</td>
<td>9</td>
<td>1.169E+06</td>
<td>256, 2048, 32, 16, 8, 32, 128, 256, 2048</td>
</tr>
<tr>
<td>17</td>
<td>0.05</td>
<td>28</td>
<td>8.052E-06</td>
<td>6</td>
<td>9.641E+05</td>
<td>256, 2048, 16, 256, 512, 512</td>
</tr>
<tr>
<td>18</td>
<td>0.5</td>
<td>19</td>
<td>8.139E-06</td>
<td>8</td>
<td>2.289E+05</td>
<td>256, 256, 32, 32, 1024, 64, 256, 128</td>
</tr>
<tr>
<td>19</td>
<td>0.005</td>
<td>14</td>
<td>8.227E-06</td>
<td>12</td>
<td>5.111E+06</td>
<td>2048, 2048, 32, 256, 16, 8, 256, 2048, 128, 64, 32, 16</td>
</tr>
<tr>
<td>20</td>
<td>0.5</td>
<td>38</td>
<td>8.338E-06</td>
<td>5</td>
<td>7.333E+05</td>
<td>128, 1024, 512, 128, 64</td>
</tr>
<tr>
<td>21</td>
<td>0.5</td>
<td>47</td>
<td>8.352E-06</td>
<td>9</td>
<td>1.646E+06</td>
<td>2048, 64, 2048, 512, 16, 64, 64, 256, 1024</td>
</tr>
<tr>
<td>22</td>
<td>0.05</td>
<td>8</td>
<td>8.481E-06</td>
<td>9</td>
<td>8.778E+05</td>
<td>512, 1024, 256, 8, 32, 1024, 64, 32, 64</td>
</tr>
<tr>
<td>23</td>
<td>0.5</td>
<td>7</td>
<td>8.489E-06</td>
<td>14</td>
<td>5.690E+05</td>
<td>1024, 64, 16, 1024, 64, 1024, 8, 64, 8, 2048, 128, 16, 16, 2048</td>
</tr>
<tr>
<td>24</td>
<td>1</td>
<td>24</td>
<td>8.501E-06</td>
<td>7</td>
<td>8.911E+05</td>
<td>128, 512, 512, 1024, 8, 8, 2048</td>
</tr>
<tr>
<td>25</td>
<td>0.05</td>
<td>15</td>
<td>8.549E-06</td>
<td>19</td>
<td>1.052E+07</td>
<td>1024, 2048, 512, 256, 2048, 1024, 32, 512, 256, 2048, 128, 2048, 64, 128, 8, 32, 2048, 1024, 1024</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>6</td>
<td>8.607E-06</td>
<td>8</td>
<td>1.386E+06</td>
<td>1024, 256, 128, 2048, 128, 32, 256, 2048</td>
</tr>
<tr>
<td>27</td>
<td>0.0005</td>
<td>6</td>
<td>8.683E-06</td>
<td>6</td>
<td>7.118E+06</td>
<td>2048, 2048, 1024, 256, 1024, 256</td>
</tr>
<tr>
<td>28</td>
<td>0.005</td>
<td>8</td>
<td>8.848E-06</td>
<td>5</td>
<td>6.305E+05</td>
<td>2048, 256, 256, 8, 256</td>
</tr>
<tr>
<td>29</td>
<td>0.005</td>
<td>34</td>
<td>8.926E-06</td>
<td>8</td>
<td>5.743E+06</td>
<td>2048, 2048, 512, 64, 512, 128, 512, 512</td>
</tr>
<tr>
<td>30</td>
<td>0.5</td>
<td>26</td>
<td>8.955E-06</td>
<td>4</td>
<td>7.164E+05</td>
<td>1024, 512, 64, 2048</td>
</tr>
</tbody>
</table>

Table S1. Top 30 models after the Step 2 HPO based on the test MSE. The validation rank in the third column is within each HPO project based on the minimum validation MSE from Step 1 HPO.
Figure S1. Validation MSE over training epochs during Step 1. Each line presents one individual tuning trial. Note that an early stopping rule is enforced with a patience of 5 epochs, while the maximum training epoch number is 100.
Figure S2. Same as Figure 2b, except the y-axis view limits.

Figure S3. Histogram of (a) the number of hidden layers and (b) the number of learnable parameters of 350 trial models (top 50 models from seven HPO projects) after retraining in Step 2. Blue (orange) bars show models with their test MSE lower than (larger than) $1 \times 10^{-2}$. 
Figure S4. Same as Figure 2c, except that each HPO project is plotted in a separate subpanel.
Figure S5. Same as Figure 2d, except that each HPO project is plotted in a separate subpanel.
Figure S6. Same as Figure 2e, except that each HPO project is plotted in a separate subpanel.

Figure S7. Inference time for 100,000 samples with a batch size of 1,024. Inference was repeated 10 times, and then, the inference times were averaged. One CPU core (AMD EPYC 7742) is used. Unit: seconds.
Figure S8. Scatter plot between Step-1 ranks (x-axis) and Step-2 ranks (y-axis) for top-50 models of each HPO project.
Figure S9. Same as Figure 2a, except that it is drawn after randomly drawn trials without replacement. The number of trials used in each panel is 25 (a), 50 (b), 100 (c), 200 (d), 400 (e), 800 (f), 1600 (g), and 10000 (h). Panel (h) is exactly identical to Figure 2a. Dashed lines denote the x value of 50.
Figure S10. Same as Figure 2c, except that it is drawn after randomly drawn trials without replacement. The number of trials used in each panel is 25 (a), 50 (b), 100 (c), 200 (d), 400 (e), 800 (f), 1600 (g), and 10000 (h). Panel (h) is identical to Figure 2c.
Figure S11. Same as Figure 2d, except that it is drawn after randomly drawn trials without replacement. The number of trials used in each panel is 25 (a), 50 (b), 100 (c), 200 (d), 400 (e), 800 (f), 1600 (g), and 10000 (h). Panel (h) is identical to Figure 2d.