Generate Neural Network Automatically with High Accuracy and High Efficiency

Xue Chao

IBM Research China
Outline

- Standard AutoML: Neural network and hyper-parameter selections
  - Bayesian optimization
  - Hyperband
  - Reinforcement learning

- Transfer AutoML
  - Using historical hyper-parameter configurations for different tasks
Standard AutoML--from Random search, Bayesian Optimization to Reinforcement Learning

Hyperparameters auto selected

<table>
<thead>
<tr>
<th>i</th>
<th>s = 4</th>
<th>s = 3</th>
<th>s = 2</th>
<th>s = 1</th>
<th>s = 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>81</td>
<td>27</td>
<td>9</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>27</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>81</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>9</td>
<td>27</td>
<td>81</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>81</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Neural Network auto selected

Adaptive Random Search (Hyperband)

Bayesian Optimization

Reinforcement Learning

$$\alpha_{\text{PI}}(x; D_n) := \mathbb{P}[v > \tau] = \Phi \left( \frac{\mu_n(x) - \tau}{\sigma_n(x)} \right)$$
• We implement the AdaRandom (adaptive random search scheme) and Reinforce (reinforcement learning search scheme) methods to generate deep learning neural network automatically.
• We are trying the new methods in different areas. Here is the example for acoustic. Default is the best scheme by manual tuning.

Neural Network:
C: Convolution, P: pooling, FC: Full connection.

Default manual: (9 layers)
C(3,32)+C(3,32)+P(3,3)+C(3,64)+C(3,64)+P(3,3)+F(64)+FC(32)+FC(6)

AdaRandom generated: (6 layers)
C(3,128)+C(4,64)+C(3,32)+P(3,3)+FC(16)+FC(6)

Reinforce generated: (5 layers)
C(3,32)+P(2,2)+P(3,2)+C(5,32)+FC(6)

The best results for the three networks are (0.703, 0.673, 0.695) (the smaller the better), that is, using AdaRandom and Reinforce recommended models, you can gain 4.3% and 1.1% in the best results comparisons. The average result of the three networks is (0.817, 0.776, 0.763), that is, the DL Insight recommended modes can increase about 5.0% and 6.6% in the average case performance. And from the standard deviation view, the recommended models are clearly more stable.

The CDF (cumulative distribution function) curve is more intuitive to illustrate the comparison of the three models (the more left the better). For example, using reinforce recommended model, ER has more than 60% probability (frequency) less than 0.75, while the default only has the 30%.
Results of AdaRandom and Bayesian Optimization for Object detection

- The raw data (no data augment)

<table>
<thead>
<tr>
<th>Machine</th>
<th>IBM P8</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>Tesla K80</td>
</tr>
<tr>
<td>GPU MEM</td>
<td>10G</td>
</tr>
<tr>
<td>CPU</td>
<td>ppc64le</td>
</tr>
<tr>
<td>CPU cores</td>
<td>160</td>
</tr>
<tr>
<td>MEM</td>
<td>55G</td>
</tr>
<tr>
<td>Frequency</td>
<td>3.5 GHz</td>
</tr>
<tr>
<td>OS</td>
<td>Ubuntu 14.04</td>
</tr>
<tr>
<td>caffe version</td>
<td>Py-faster-rcnn</td>
</tr>
<tr>
<td>Dataset</td>
<td>User-defined</td>
</tr>
<tr>
<td>Model</td>
<td>vgg16</td>
</tr>
</tbody>
</table>

- The average results for the three (HPP) hyperparameters combinations at 4000 iterations are (0.49, 0.52, 0.55), that is, using AdaRandom and Bayesian optimization recommended HPP, you can gain 6% and 12% in the average results comparing to default setting. AdaRandom method has more variance among different tries (train-test dataset split). The Bayesian models are more stable while having better average performance.

- The below pictures show accuracy during the 0-4000 iterations, with different tries, under the two HPP configurations. We can see that: 1) It can be early stopped at about 1300 iterations. 2) the performance with different tries differ significantly, it caused by in some tries, training dataset has the invariance property according to test dataset, but some doesn’t have. It need to augment data to gain the stable performance. 3) different HPP combinations(models) may have different sensitivity to the different tries.
We implement these three standard methods by using design pattern of strategy whose UML is shown:

We implemented the system as the server-client model: the server is responsible for generating the candidate neural network, and scheduling the tasks to the clients. The clients focused on calculating the accuracy of the allocated neural networks. The tasks in the clients can be executed in many frameworks, such as Caffe, TF, which are transparent to the server.
Transfer AutoML framework

Weights and bias

Neural network

Hyper-parameter

Dataset group

Dog

Car

Unknown

Parameter

Model

Virtual Dataset

Dataset

Traditional fine-tune

Standalone AutoML

Collaborative AutoML

Transfer AutoML framework

Weights and bias

Neural network

Hyper-parameter

Dataset group

Dog

Car

Unknown

Parameter

Model

Virtual Dataset

Dataset

Traditional fine-tune

Standalone AutoML

Collaborative AutoML
Transfer AutoML Architecture

Upload datasets → White-box analysis → Benchmark models selection → Black-box analysis

Knowledge base → Virtual datasets group

Path 1: Parallelization A → Server (RL, HyperBand) → Clients
Path 2: Joint Optimization with transfer learning → Model selection

AutoML process:
- Server (RL, HyperBand)
- Clients (Spark or others)

© 2013 IBM Corporation
Challenges and solutions in AutoML

- Short responding time Vs. long time to do AutoML
  - High efficient AutoML
    - Used share information among tasks
  - Parallelization B is important to speed up AutoML
    - Design paralleled AutoML algorithms

- Large datasets and deep neural network
  - Requires huge resources using AutoML for large datasets and deep neural network
    - Reduce the searching space methods
    - Parallelization B is important to split large datasets and deep neural networks into calculable size in one machine
Challenges for AutoML with Transfer Learning

- Training small user dataset leads to convergence problem → **Transfer learning** is needed
- When considering transfer learning, the **pretrained model** need to be chosen, usually in the computer vision, we choose **image-net** as the base dataset to get the initial weights as the pretrained model, but it *can’t fit many specific user datasets*.
- To solve this transfer learning’s problem, we can let the user to classify his dataset into some **predefined categories**, and in each category, the pretrained model was trained separately. It can improve the performance of transfer learning but **involve user’s intervention** with their datasets.
- Using **AutoML** with transfer learning can improve transfer learning’s performance without user’s intervention. But considering the transfer learning’s properties, there are **two challenges** for AutoML:
  – Since reusing the initial weights, transfer learning **limits the searching space** of AutoML, how to **use AutoML based on the pretrained model** is a question.
  – We can’t use AutoML to build one model for every user dataset, it is too expensive. How to **reuse the model** for transfer learning is a question.
Joint optimization: AutoML with the fine-tune
Standard AutoML Used for Offline Stage

Search space:

1) Neural network at the last stage:
   Example: 

```
layer {
  name: "last_fc"
  type: "InnerProduct"
  bottom: "pool1"
  top: "last_fc"
  param {
    lr_mult: LR_MULT_FC_W_0
    decay_mult: DECAY_MULT_FC_W_0
  }
  param {
    lr_mult: LR_MULT_FC_B_0
    decay_mult: DECAY_MULT_FC_B_0
  }
}
```

2) Below hyper-parameter

- lr_policy: LR_POLICY
- stepsize: STEPSIZE
- gamma: GAMMA
- momentum: MOMENTUM
- solver_mode: GPU
- max_iter: MAX_ITER
- test_iter: TEST_ITER
- test_interval: TEST_INTERVAL
- base_lr: BASE_LR
- weight_decay: WEIGHT_DECAY
- solver_type: SGD

```
layer {
  name: "conv0_relu"
  type: TYPE_C_AF_0
  bottom: "conv0"
  top: "conv0"
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv0"
  top: "pool1"
  pooling_param {
    pool: AVE
    kernel_size: KERNEL_SIZE_P_0
    stride: STRIDE_P_0
  }
}
```

Search space:

1) Neural network at the last stage:
   Example: 

```
layer {
  name: "last_fc"
  type: "InnerProduct"
  bottom: "pool1"
  top: "last_fc"
  param {
    lr_mult: LR_MULT_FC_W_0
    decay_mult: DECAY_MULT_FC_W_0
  }
  param {
    lr_mult: LR_MULT_FC_B_0
    decay_mult: DECAY_MULT_FC_B_0
  }
}
```

2) Below hyper-parameter

- lr_policy: LR_POLICY
- stepsize: STEPSIZE
- gamma: GAMMA
- momentum: MOMENTUM
- solver_mode: GPU
- max_iter: MAX_ITER
- test_iter: TEST_ITER
- test_interval: TEST_INTERVAL
- base_lr: BASE_LR
- weight_decay: WEIGHT_DECAY
- solver_type: SGD
Advantages of AutoML with finetune:
(+) Best accuracy
(+) Most stable
(+) Don’t need separate pretrained models by predefining dataset categories. No user’s interventions.

Disadvantages of AutoML with finetune:
(-) Overhead cost of running benchmark models. But we can shorten the process to about 2 minutes by good design.
Reference


Thank you for listening