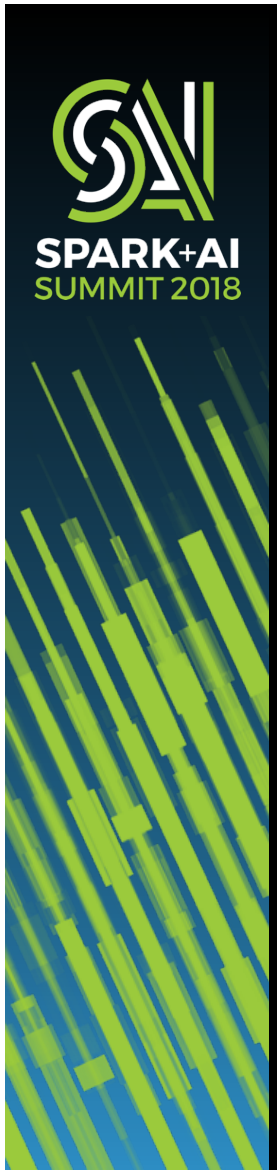


Hyper-Parameter Selection and Adaptive Model Tuning for Deep Neural Networks

YongGang Hu, Chao Xue, IBM

#AssignedHashtagGoesHere

Outline

- Hyper-parameter selections & Neural network search
  - Bayesian optimization
  - Hyperband
  - Reinforcement learning
- Transfer AutoML
  - Using historical hyper-parameter configurations for different tasks
  - Joint optimization with AutoML and finetune
- Training visualization and Interactive tuning
- Real world experience and Implementation

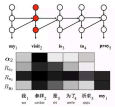
From Hyper parameter/Network Search to On-line Tuning

Deep Learning Applications

Image recognition

Object detection

Translation



Others

Optimize

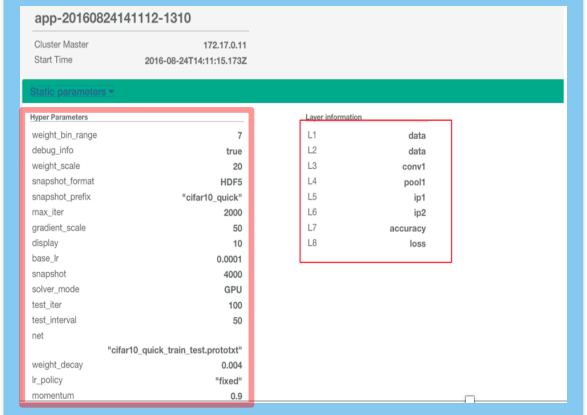
Hyper-parameter & Network Search

optimizing procedure parameters

example: learning rate, batch size, momentum, learning rate scheme

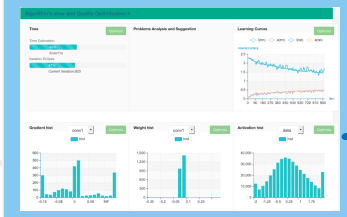
criterion and model parameters

example: number of hidden units, number of layers, filter kernel size, max-pool size, weight decay



Monitor

Real-time monitor for running application



learning curve

weight/gradient/activation histogram and norm

worst cases of training samples

Advice

Real-time adviser for running application

check the training process

overflow

underfitting

overfitting

divergence

convergence

Standard AutoML--from Random search, Bayesian Optimization to Reinforcement Learning

Hyperparameters auto selected

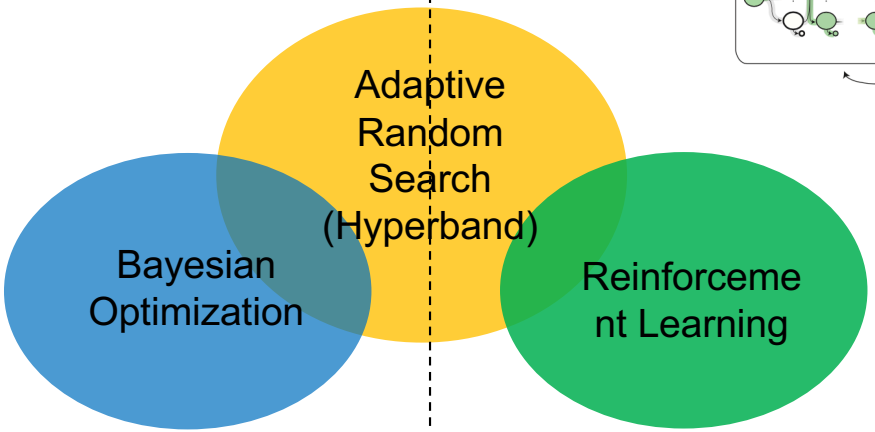
	$s=4$		$s=3$		$s=2$		$s=1$		$s=0$	
i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i	n_i	r_i
0	81	1	27	3	9	9	6	27	5	81
1	27	3	9	9	3	27	2	81		
2	9	9	3	27	1	81				
3	3	27	1	81						
4	1	81								

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix}.$$

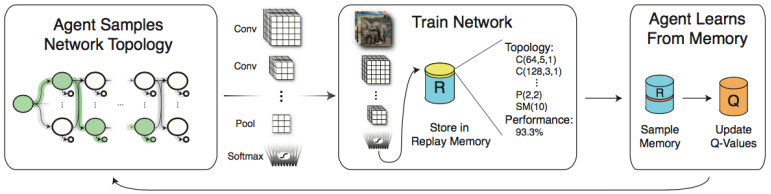
$$\begin{aligned} \boldsymbol{\mu}_{a|b} &= \boldsymbol{\mu}_a + \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}(\mathbf{x}_b - \boldsymbol{\mu}_b) \\ \boldsymbol{\Sigma}_{a|b} &= \boldsymbol{\Sigma}_{aa} - \boldsymbol{\Sigma}_{ab}\boldsymbol{\Sigma}_{bb}^{-1}\boldsymbol{\Sigma}_{ba}. \end{aligned}$$

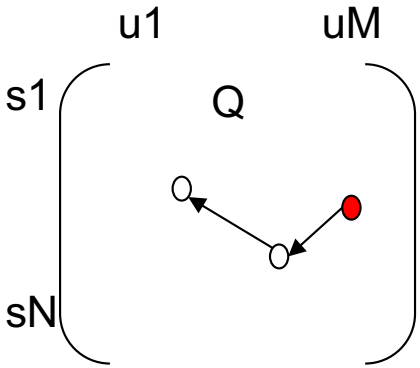
$$\begin{aligned} \mu_n(\mathbf{x}) &= \mu_0(\mathbf{x}) + \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}(\mathbf{y} - \mathbf{m}) \\ \sigma_n^2(\mathbf{x}) &= k(\mathbf{x},\mathbf{x}) - \mathbf{k}(\mathbf{x})^T(\mathbf{K} + \sigma^2\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}), \end{aligned}$$

$$\alpha_{\mathrm{PI}}(\mathbf{x}; \mathcal{D}_n) := \mathbb{P}[v > \tau] = \Phi\left(\frac{\mu_n(\mathbf{x}) - \tau}{\sigma_n(\mathbf{x})}\right)$$



Neural Network auto selected

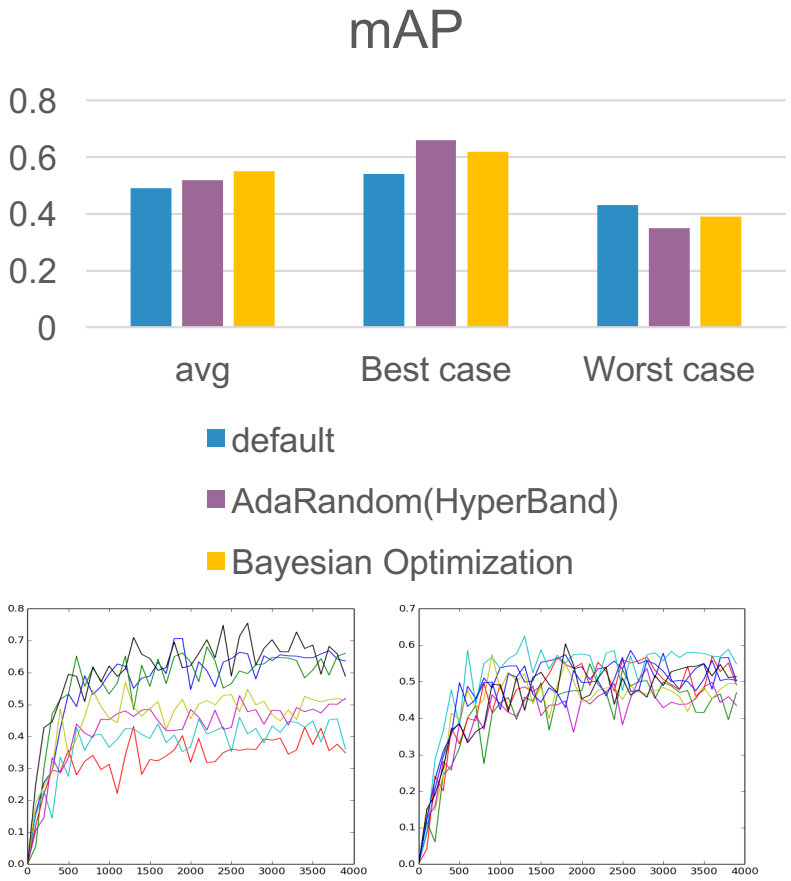




Results of AdaRandom and Bayesian Optimization for Object detection

- The raw data (no data augment)

Machine	IBM P8
GPU	Tesla K80
GPU MEM	10G
CPU	ppc64le
CPU cores	160
MEM	55G
Frequency	3.5 GHz
OS	Ubuntu 14.04
caffe version	Py-faster-rcnn
Dataset	User-defined
Model	vgg16



- 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.

Evaluations with Acoustic Applications

- 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.

Lower complexity

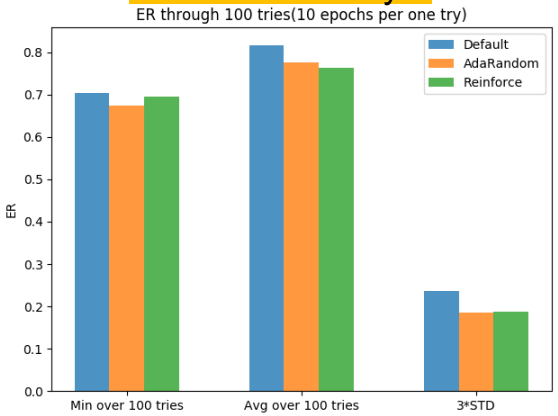
Neural Network:
C: Convolution, P: pooling, FC: Full connection.

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

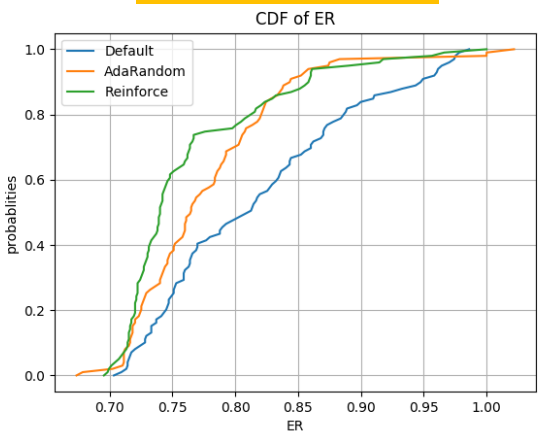
AdaRandom generated: (6 layers)
C(3,128)+C(4,64)+C(3,32)+P(3,3)+FC(16)+FC(6)

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

Better accuracy

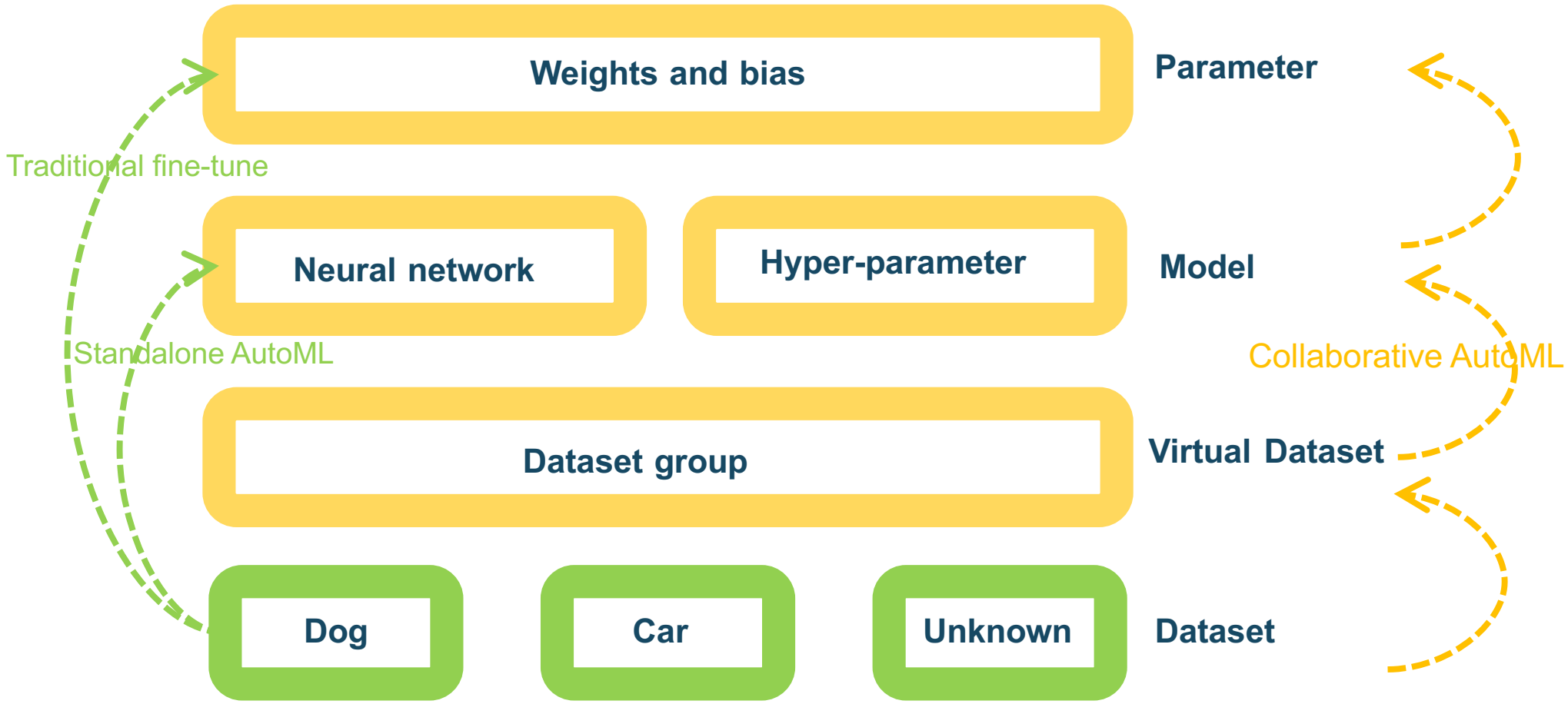


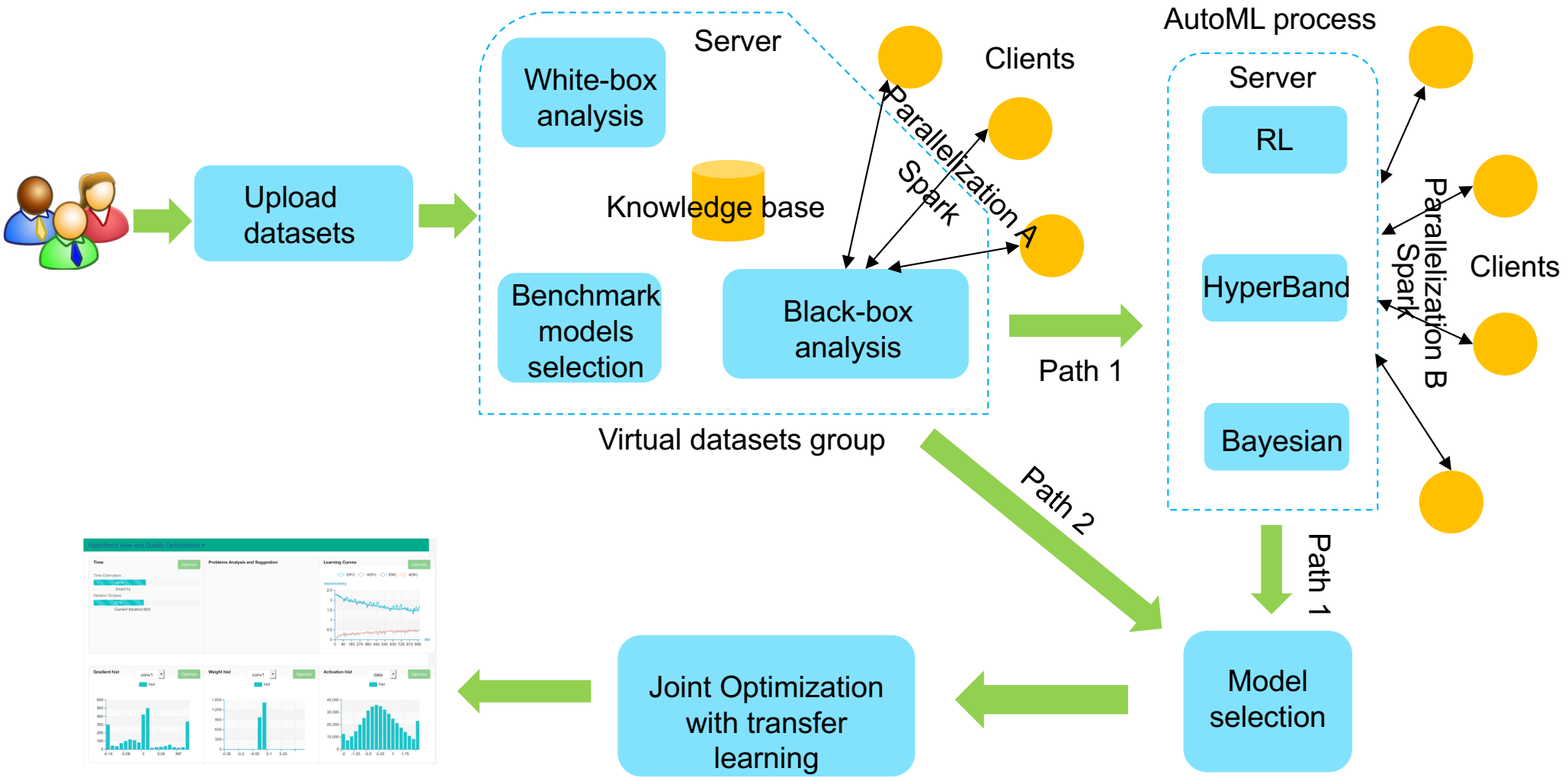
More stable



- 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%**.

Transfer AutoML Architecture

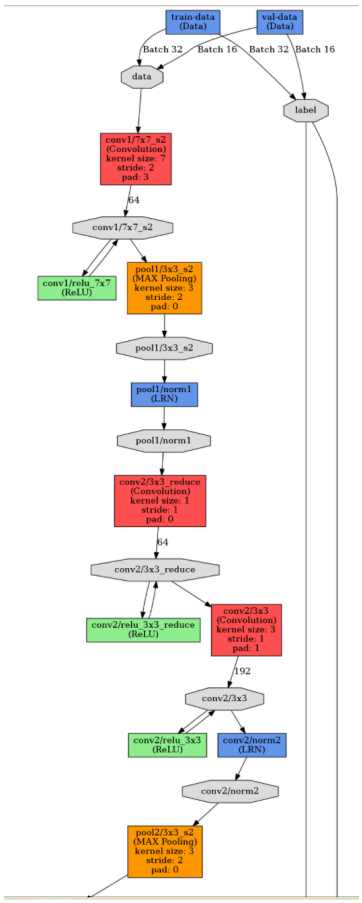


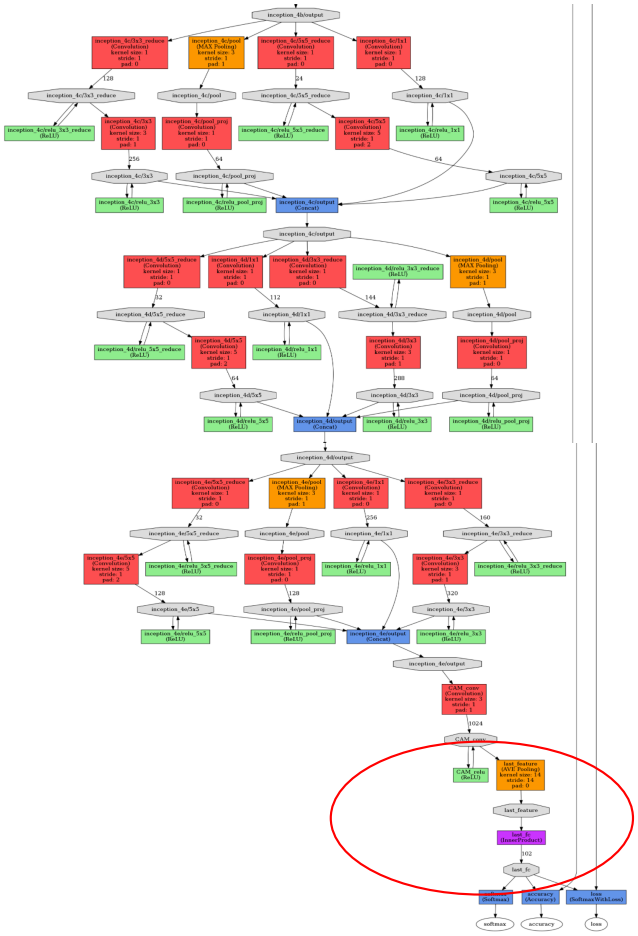

Upload datasets
Server
White-box analysis
Knowledge base
Benchmark models selection
Black-box analysis
Clients
Parallelization A
Spark
Virtual datasets group
Path 1
AutoML process
Server
RL
HyperBand
Bayesian
Parallelization B
Spark
Clients
Path 2
Path 1
Model selection
Joint Optimization with transfer learning

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





Search space:

1) Neural network at the last stage:
   Example:

2) Below hyper-parameter

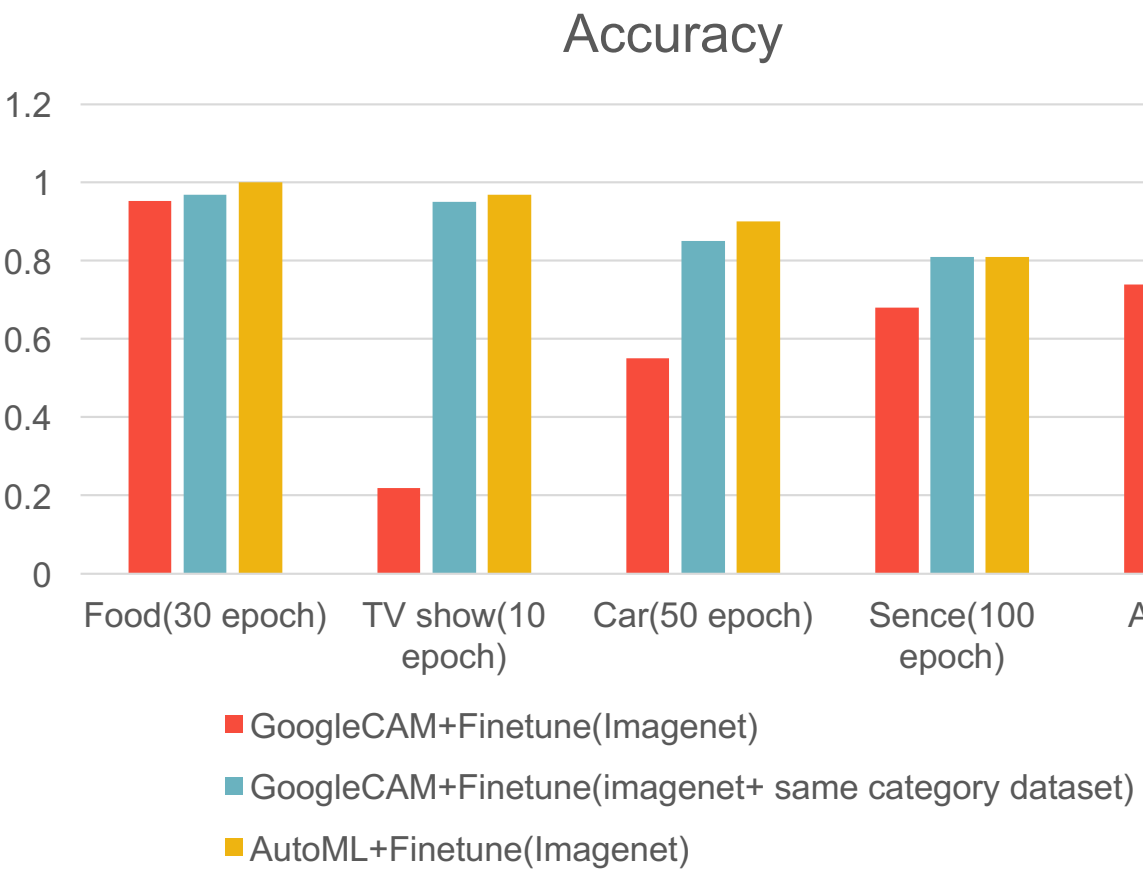
- lr_policy: LR_POLICY
  stepsize: STEPSIZE
  gamma: GAMMA
  momentum: MOMENTUM
  solver_mode: GPU
  max_iter: MAX_ITER
  test_iter: TEST_ITER
  test_interval: TEST_INTERVAL
  base_lr: BASE_LR
  weight_decay: WEIGHT_DECAY
  solver_type: SGD

```
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
  }
}
```

```
layer {
  param {
    lr_mult: LR_MULT_C_W_0
    decay_mult:
DECAY_MULT_C_W_0
  }
  param {
    lr_mult: LR_MULT_C_B_0
    decay_mult:
DECAY_MULT_C_B_0
  }
  convolution_param {
    num_output: NUM_OUTPUT_0
    pad: PAD_0
    kernel_size: KERNEL_SIZE_C_0
    group: GROUP_0
    weight_filler {
      type: TYPE_C_W_0
      std: STD_C_W_0
    }
    bias_filler {
      type: TYPE_C_B_0
      std: STD_C_B_0
    }
  }
}
```

```
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
  }
  inner_product_param {
    num_output: OUTPUT_NUMS
    weight_filler {
      type: TYPE_FC_W_0
      std: STD_FC_W_0
    }
    bias_filler {
      type: TYPE_FC_B_0
      std: STD_FC_B_0
    }
  }
}
```

Results and Analysis



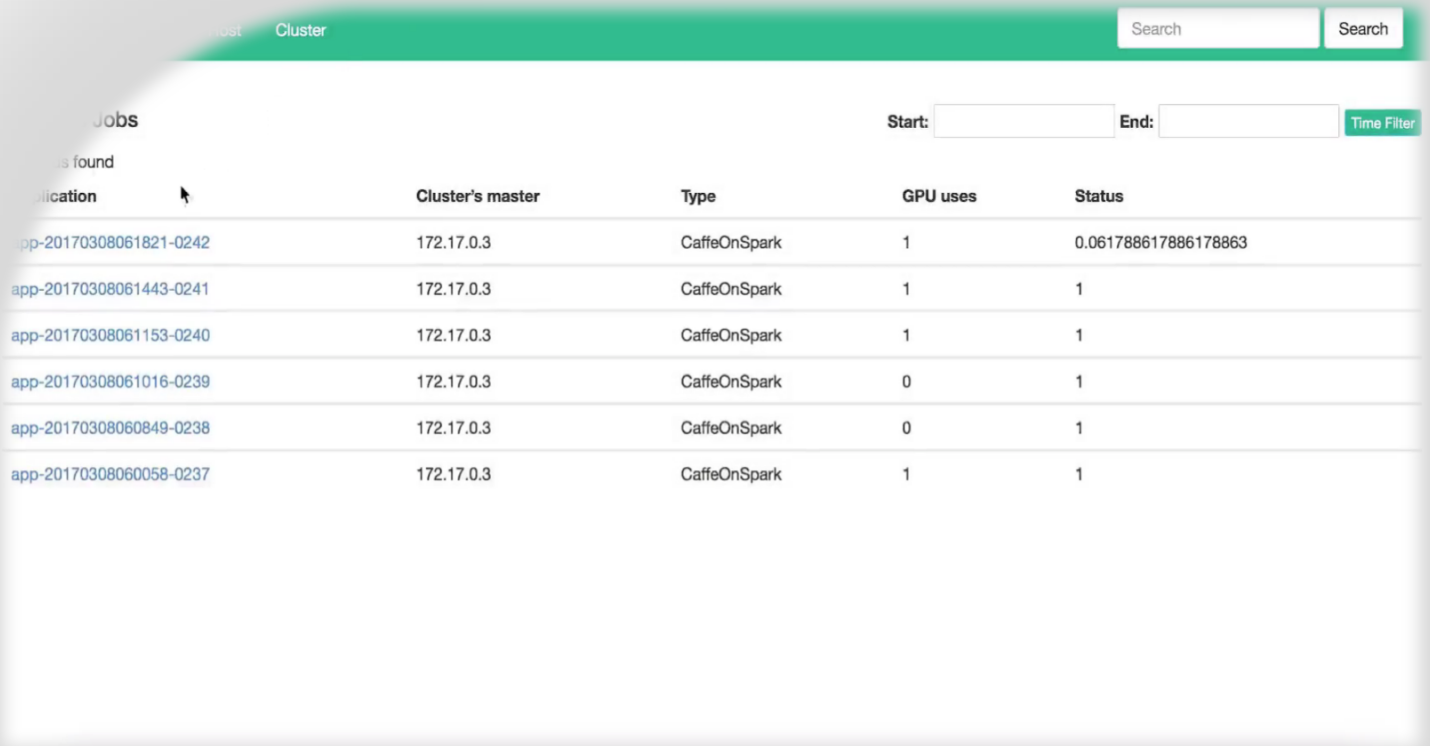
Advantages of AutoML with finetune:
(+) Best accuracy
(+) Most stable
(+) Don't need separate pretrained models by predefining dataset categories. No user's interventions.

Training Monitoring & Interactive Tuning of Hyper Parameters

Expert optimization advice for hyper parameter selection and tuning

Traffic light alerting for required parameter optimization with early stop advice and more

CPU, GPU, memory utilization info, comms overhead, +++

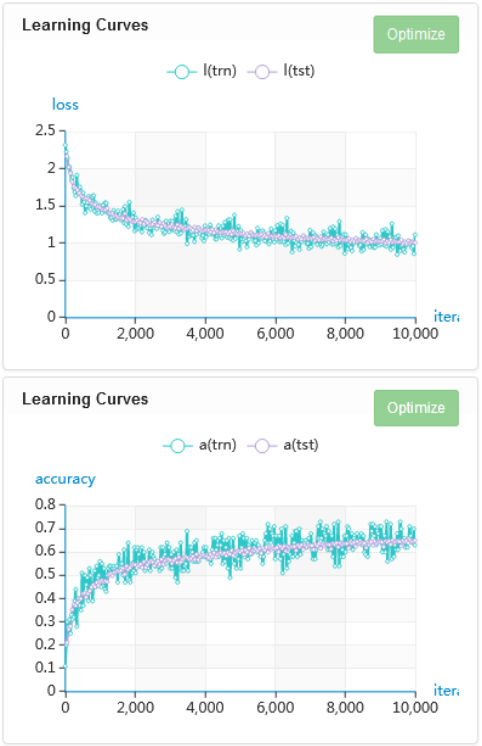


Auto-detection method for training process

CIFAR-10

Traditional method: underfitting

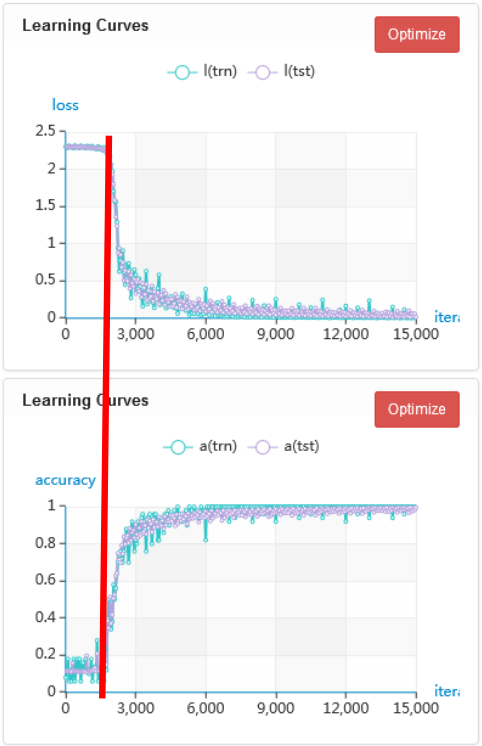
Our method: still going down



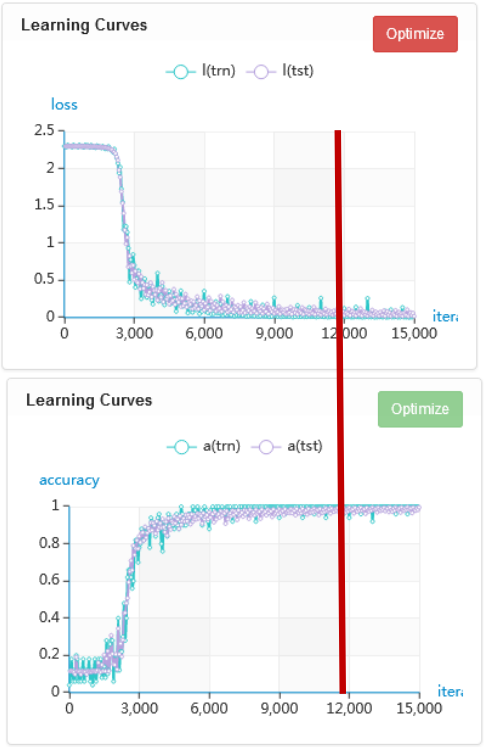
Auto-detection method for training process

MNIST

Traditional method: underfitting



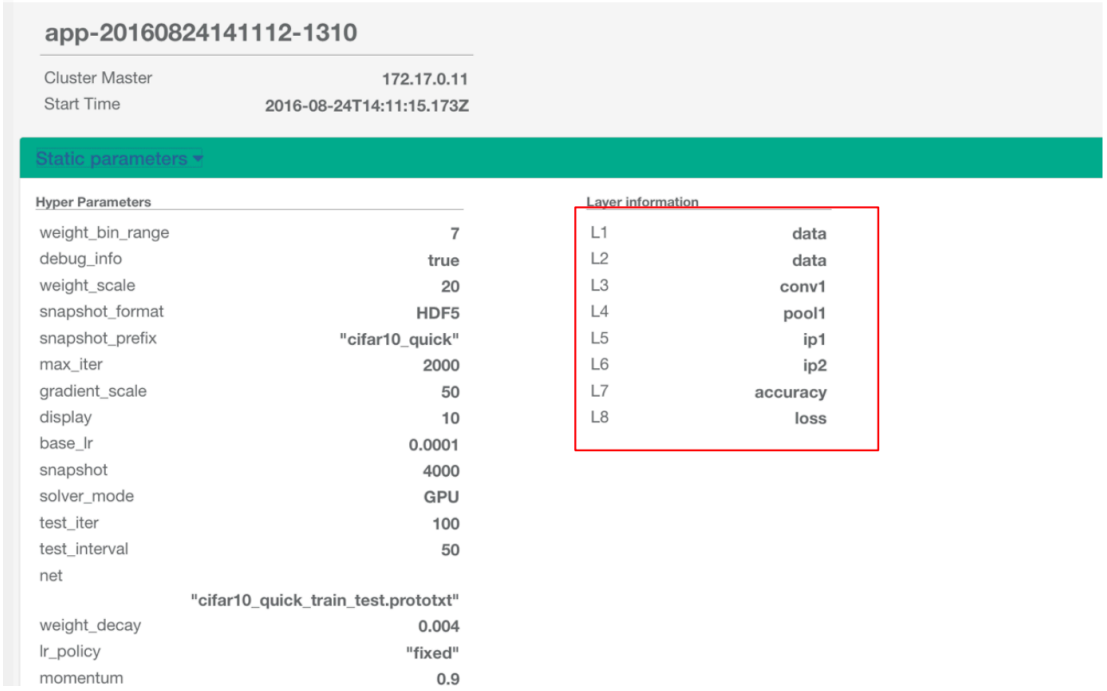
Our method: good game



Interactive Tuning – Example - 1st run of Caffe

CIFAR-10

- We will run three times of caffe on spark, to show MAO's capabilities for DL.
  - 1st run of caffe on spark using cifar-10



app-20160824141112-1310

Cluster Master	172.17.0.11
Start Time	2016-08-24T14:11:15.173Z

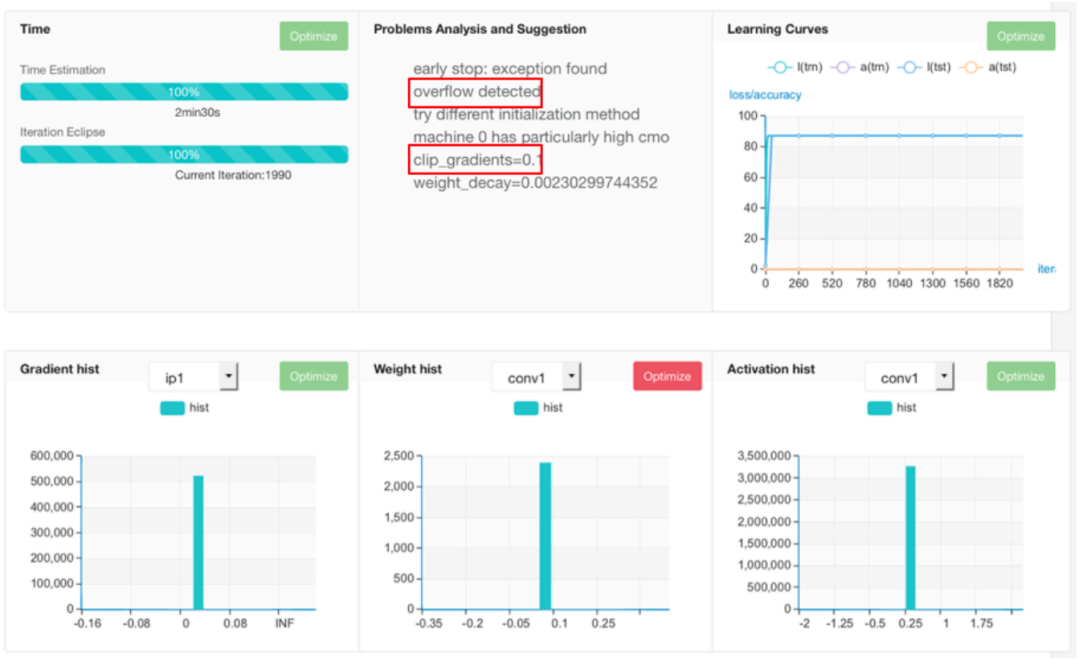
Static parameters

Hyper Parameters	
weight_bin_range	7
debug_info	true
weight_scale	20
snapshot_format	HDF5
snapshot_prefix	"cifar10_quick"
max_iter	2000
gradient_scale	50
display	10
base_lr	0.0001
snapshot	4000
solver_mode	GPU
test_iter	100
test_interval	50
net	"cifar10_quick_train_test.prototxt"
weight_decay	0.004
lr_policy	"fixed"
momentum	0.9

Layer information	
L1	data
L2	data
L3	conv1
L4	pool1
L5	ip1
L6	ip2
L7	accuracy
L8	loss

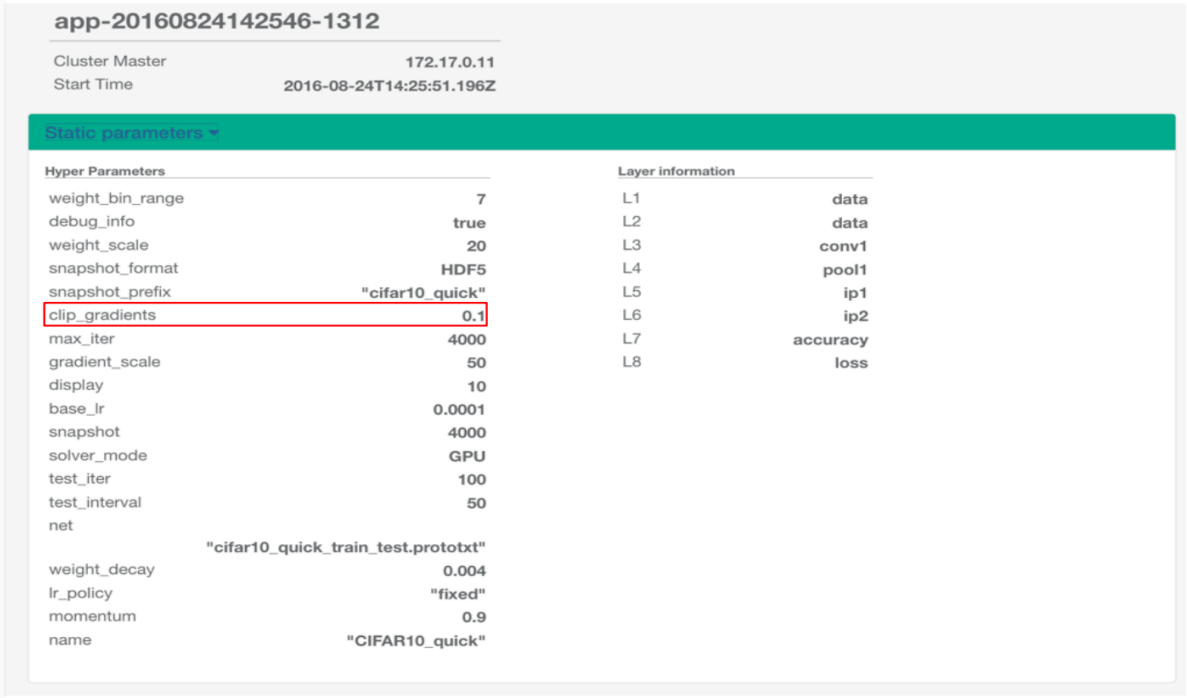
- The caffe model is 8 layers network as on the right, the dataset is cifar-10, the type is caffe on spark, and the hyperparameter is on the left.

Interactive Tuning – Example - 1st run of Caffe



- The weight histogram finds the problem that some weights are nan.
- The MAO suggests user to stop the training early because overflow detected, and give the user advisor to add clip gradient to 0.1

Interactive Tuning – Example - 2nd run of Caffe



app-20160824142546-1312

Cluster Master	172.17.0.11
Start Time	2016-08-24T14:25:51.196Z

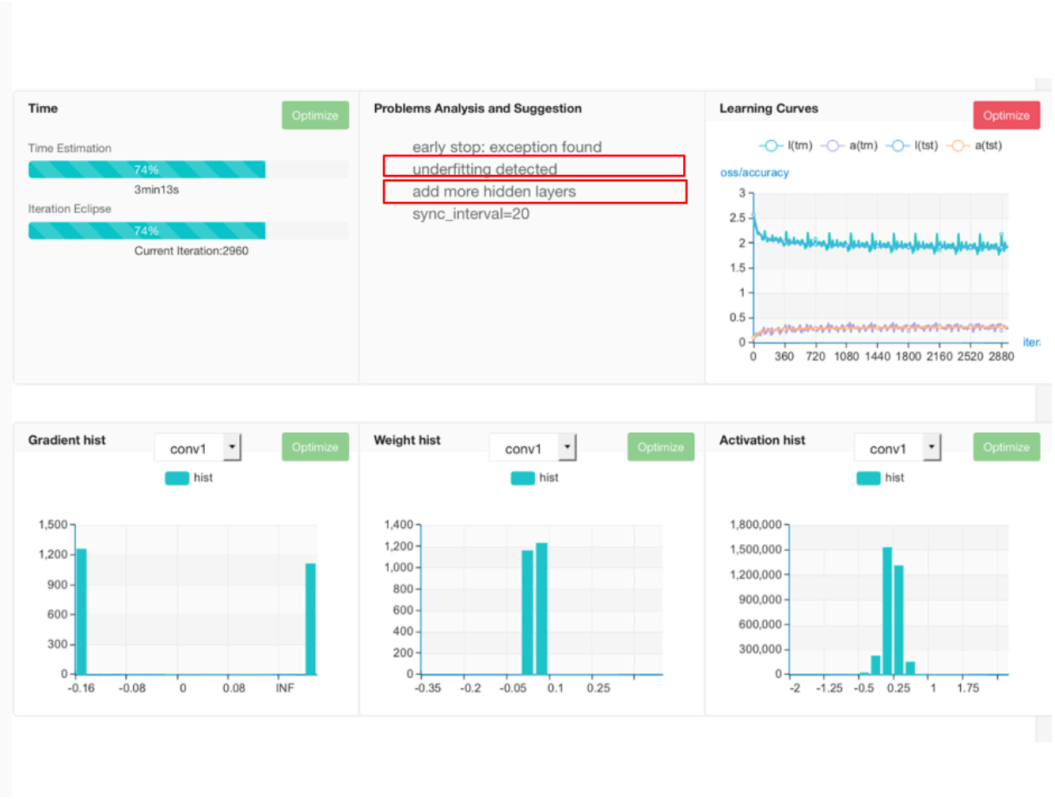
Static parameters

Hyper Parameters	
weight_bin_range	7
debug_info	true
weight_scale	20
snapshot_format	HDF5
snapshot_prefix	"cifar10_quick"
clip_gradients	0.1
max_iter	4000
gradient_scale	50
display	10
base_lr	0.0001
snapshot	4000
solver_mode	GPU
test_iter	100
test_interval	50
net	"cifar10_quick_train_test.prototxt"
weight_decay	0.004
lr_policy	"fixed"
momentum	0.9
name	"CIFAR10_quick"

Layer information	
L1	data
L2	data
L3	conv1
L4	pool1
L5	ip1
L6	ip2
L7	accuracy
L8	loss

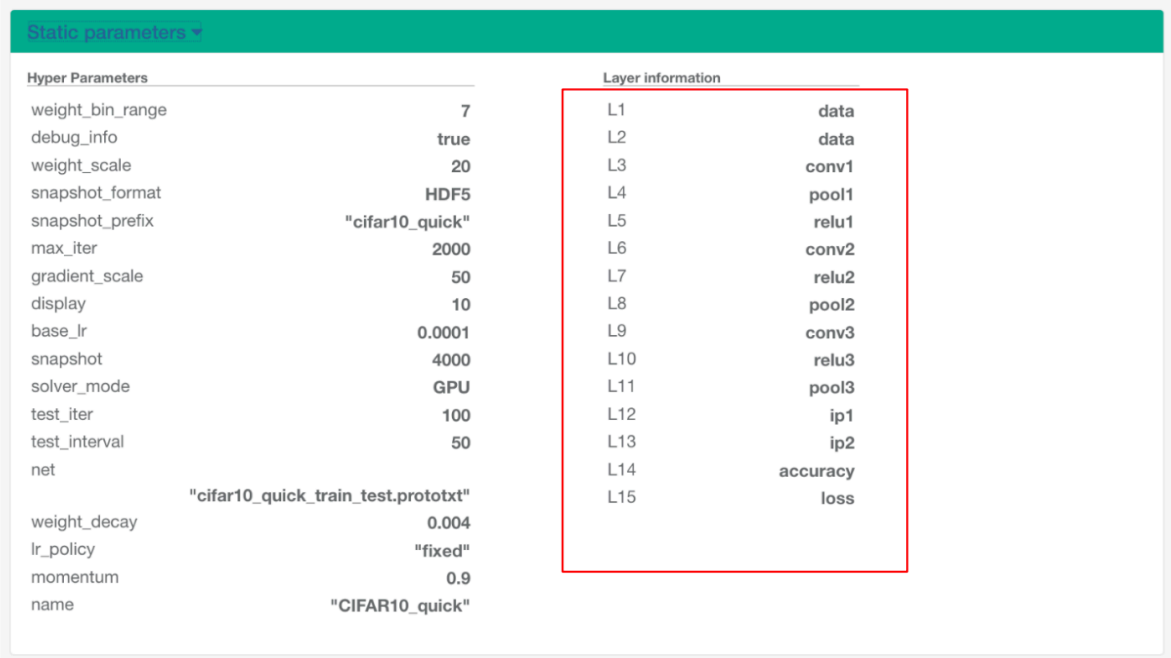
- On the 2nd run of caffe, we accept the suggestion which MAO gave, and add clip gradient to 0.1, what's happened then?

Interactive Tuning – Example - 2nd run of Caffe



- The nan of weights is disappeared, and the overflow error is away.
- The learning curves find the problem that the curve is converged but the accuracy is still so low about 0.3 on both training and testing.
- The MAO suggests user to stop the training early because underfitting detected, and give the user advisor to add more hidden layers.

Interactive Tuning – Example -3rd run of Caffe



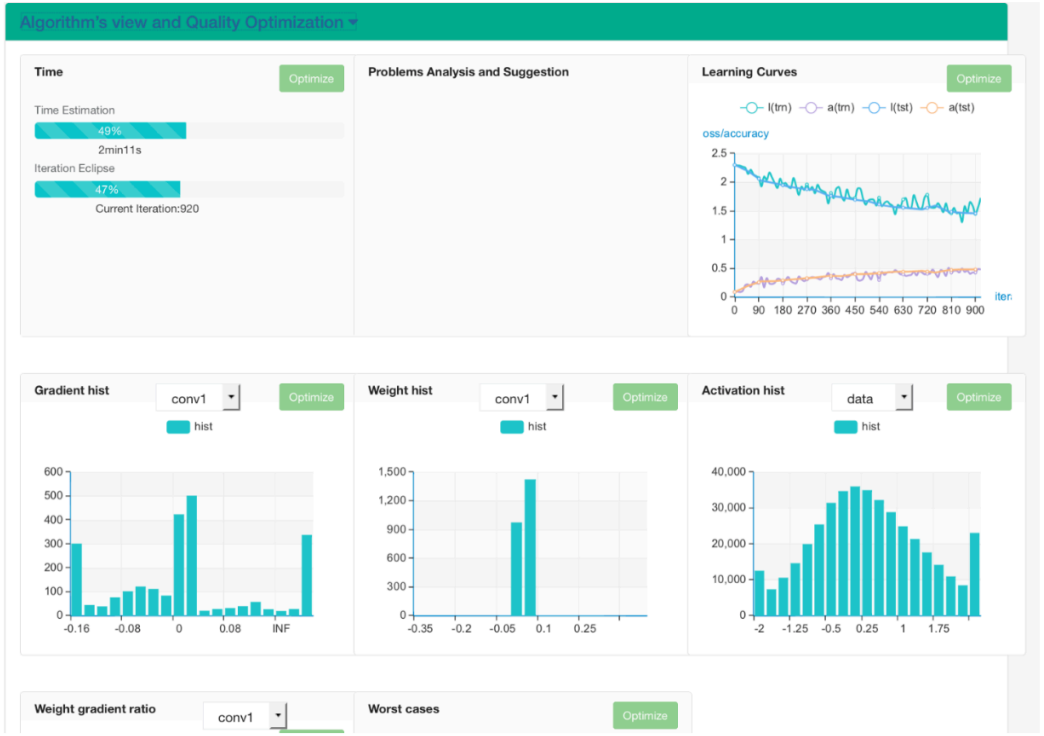
Static parameters

Hyper Parameters	
weight_bin_range	7
debug_info	true
weight_scale	20
snapshot_format	HDF5
snapshot_prefix	"cifar10_quick"
max_iter	2000
gradient_scale	50
display	10
base_lr	0.0001
snapshot	4000
solver_mode	GPU
test_iter	100
test_interval	50
net	"cifar10_quick_train_test.prototxt"
weight_decay	0.004
lr_policy	"fixed"
momentum	0.9
name	"CIFAR10_quick"

Layer information	
L1	data
L2	data
L3	conv1
L4	pool1
L5	relu1
L6	conv2
L7	relu2
L8	pool2
L9	conv3
L10	relu3
L11	pool3
L12	ip1
L13	ip2
L14	accuracy
L15	loss

On the 3rd run of caffe, we accept the suggestion which MAO gave, and more layers such as RELU, what's happened then?

Interactive Tuning – Example - 3rd run of Caffe

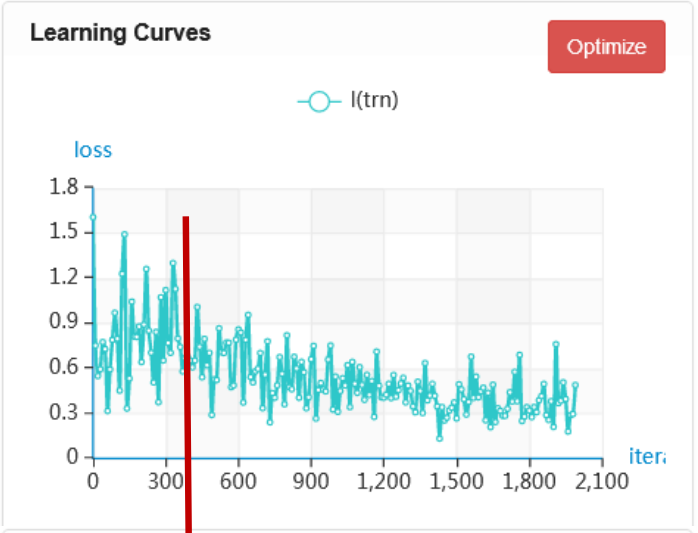


- Everything looks well, the accuracy get about 0.5 after just 900 iterations, and that is still growing.

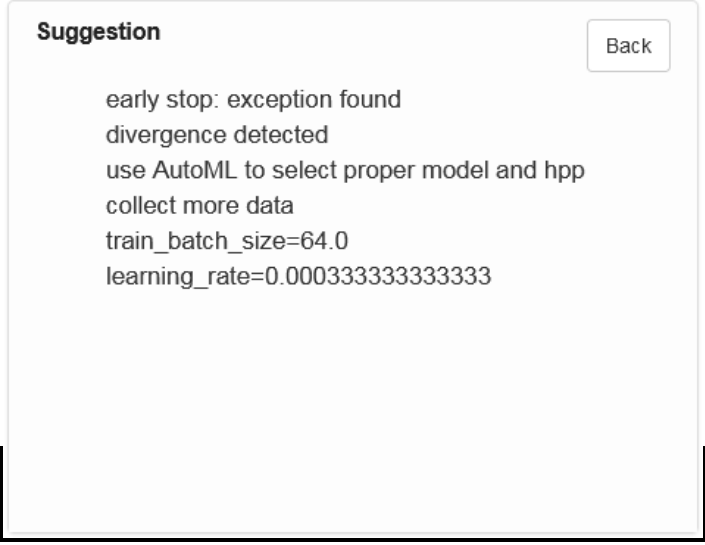
- No suggestion MAO gave to you, you can do other things or have a rest while training the dataset. When you feel worried about it, just have a look at MAO.🙂

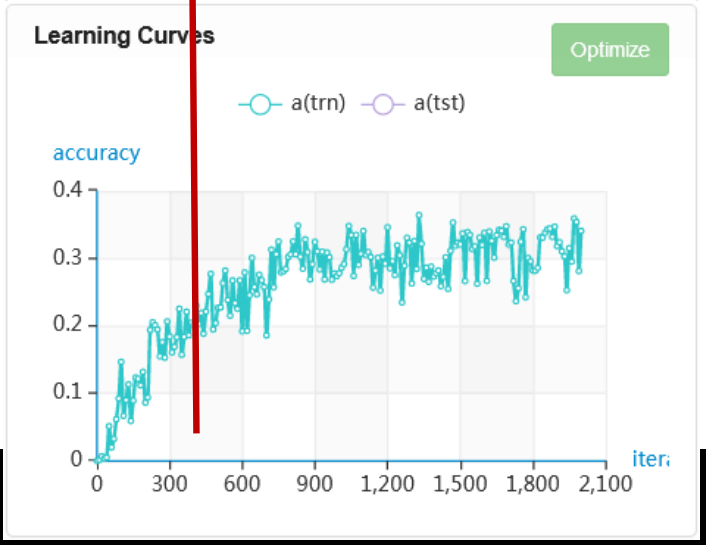
Interactive Tuning - Example: “Applause” datasets

runs	Hyperparameters in suggestions					Parameters			AP	mAP	mAP at early stop iteration
	base_lr	weight_decay	momentum	early stop	batch size/rpn batch size	split ratio	max_iters	display			
applause01	0.001	0.0005	0.9	yes divergence	128/256	0.7(94/39)	2000	10	appl = 0.3409	0.3409	iter=370 mAP=0.221060



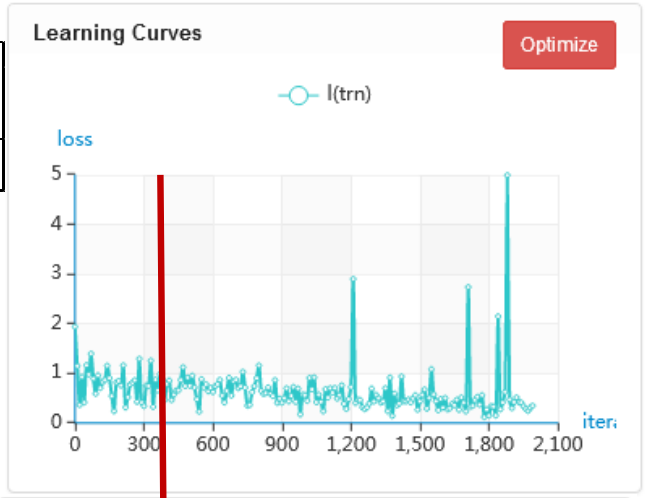
Good judgement

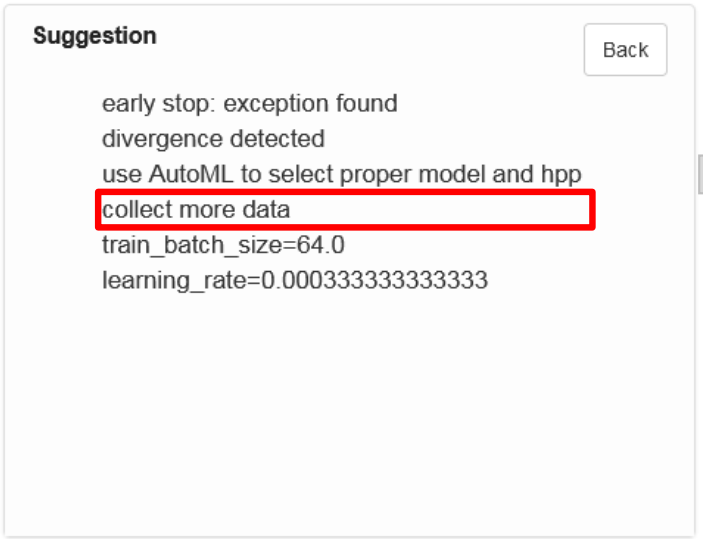




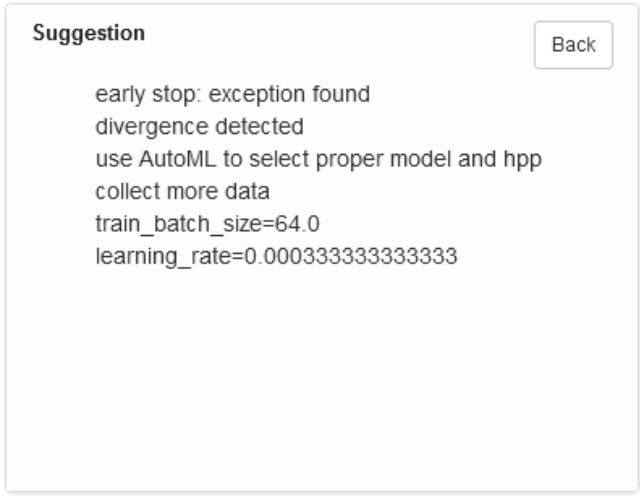
Learning Curve Trend Judgement – 'Applause' (Continue)

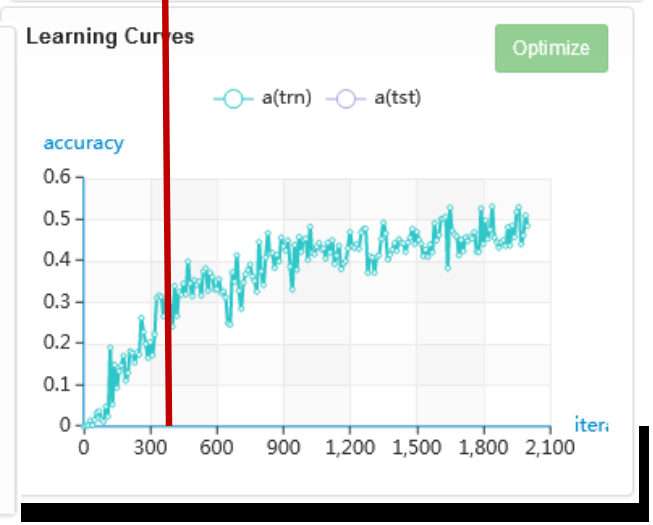
runs	Hyperparameters in suggestions					Parameters			AP	mAP	mAP at early stop iteration
	base_lr	weight_decay	momentum	early stop	batch size/rpn batch size	split ratio	max_iters	display			
applause-aug01-01	0.001	0.0005	0.9	yes divergence	128/256	0.7(98/42)	2000	10	AP for appl = 0.4848	0.4848	iter=350 mAP=0.313152





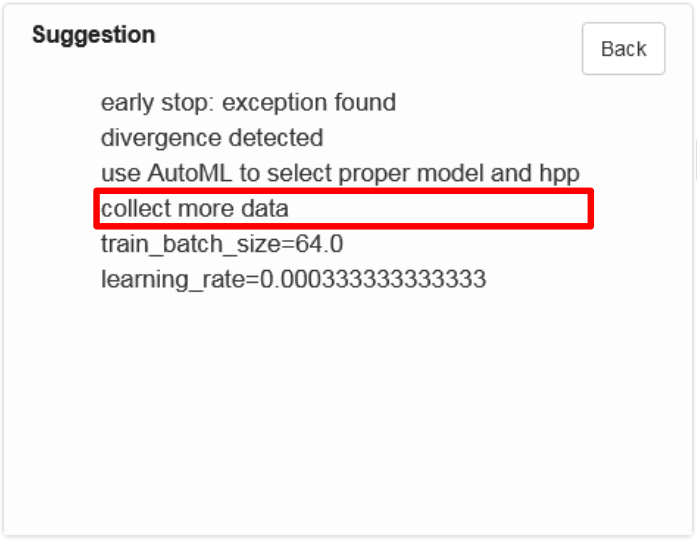
Good judgement



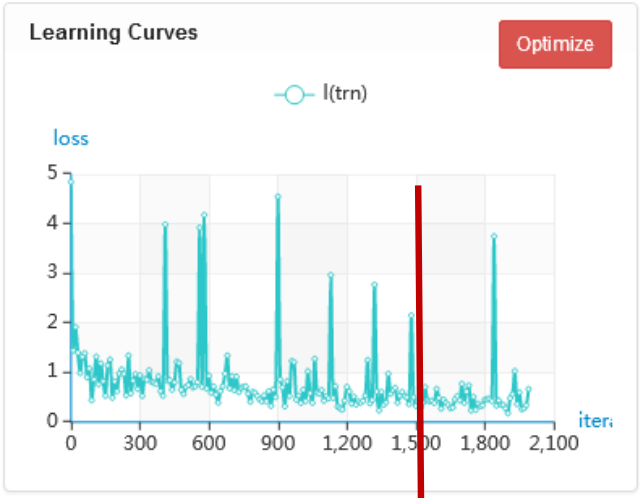


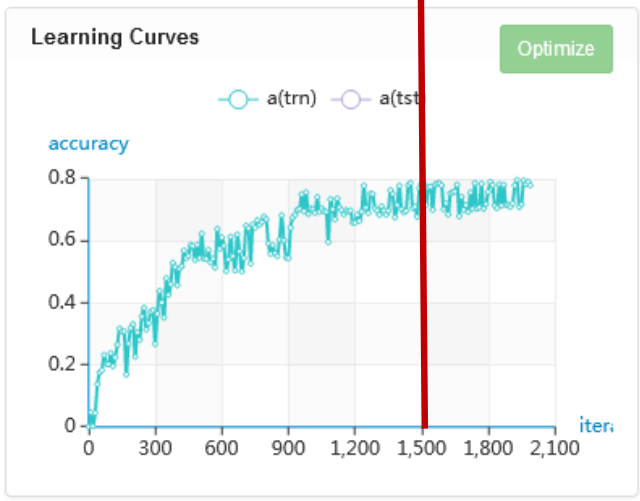
Learning Curve Trend Judgement – 'Applause' (Continue)

runs	Hyperparameters in suggestions					Parameters			AP	mAP	mAP at early stop iteration
	base_lr	weight_decay	momentum	early stop	batch size/rpn batch size	split ratio	max_iters	display			
applause-aug03-01	**0.001**	**0.0005**	**0.9**		**128/256**	**0.7(164/69)**	**2000**	**10**	AP for appl = 0.7847	**0.7847**	iter=1510 mAP=0.784068



Good judgement



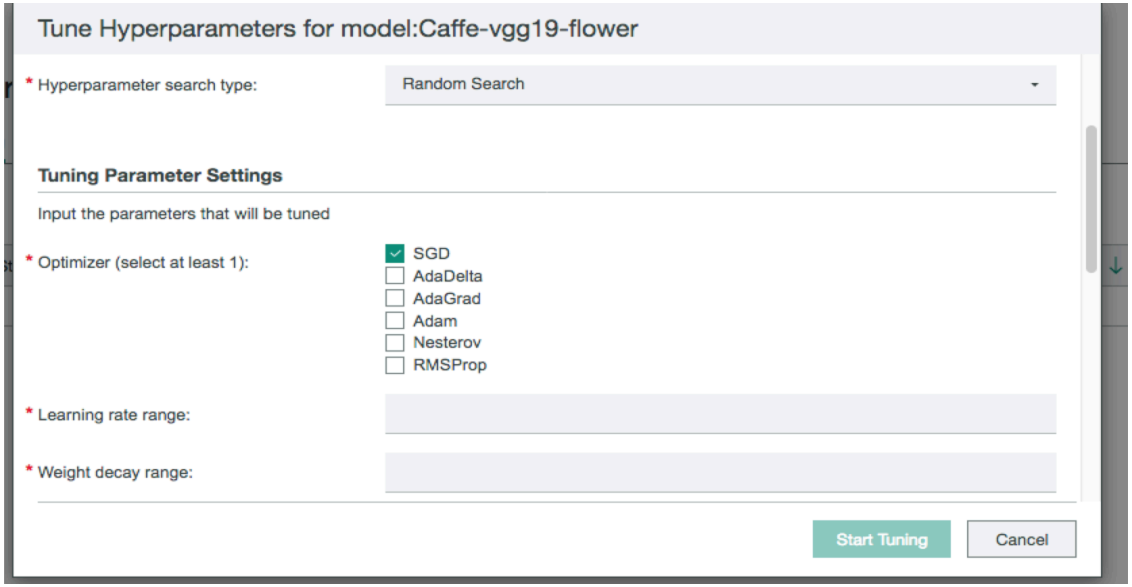


Hyper-Parameter Search Implementation

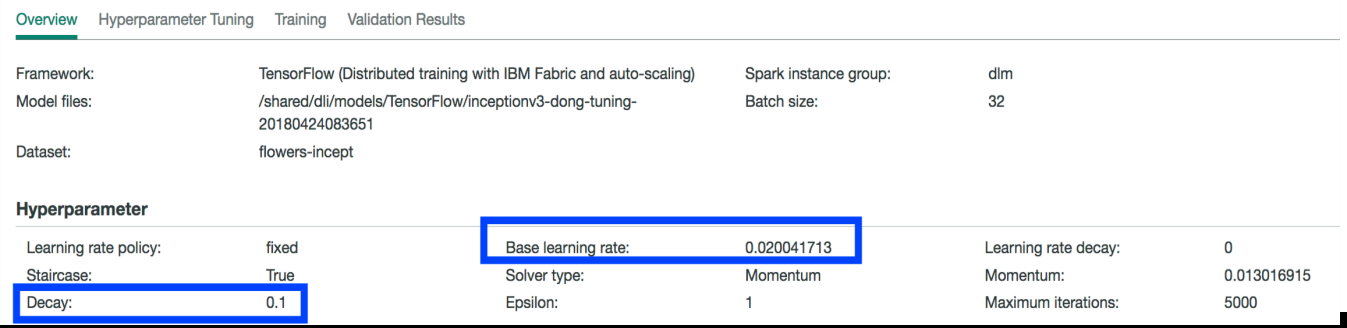
IBM Spectrum Conductor

Search hyper-parameters space :

- Learning rate
- Decay rate
- Batch size
- Optimizer:
  - GradientDecedent,
  - Adadelta,
  - …
- Momentum (for some optimizers)
- LSTM hidden unit size

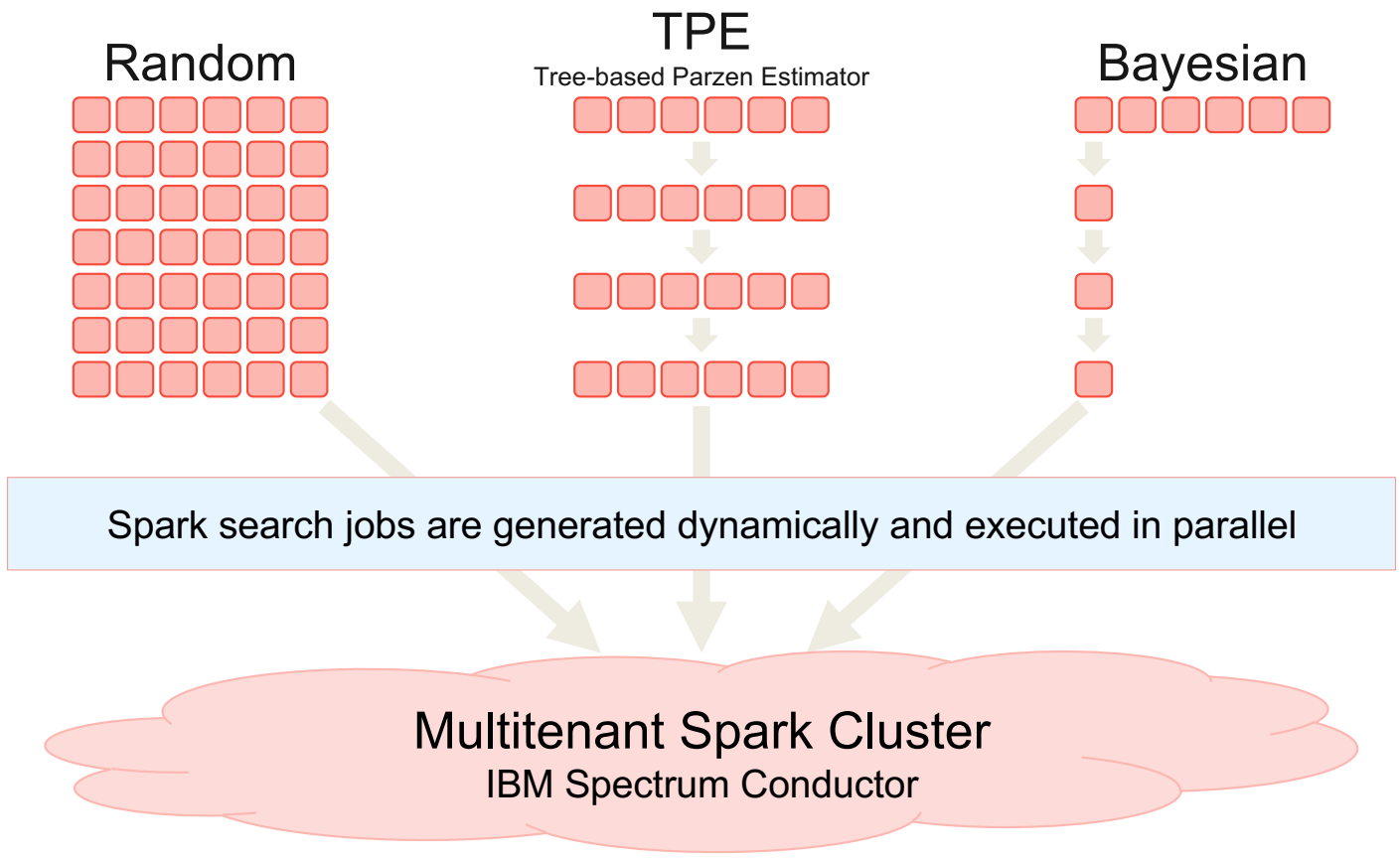


Random, Bayesian, TPE Based Search Types



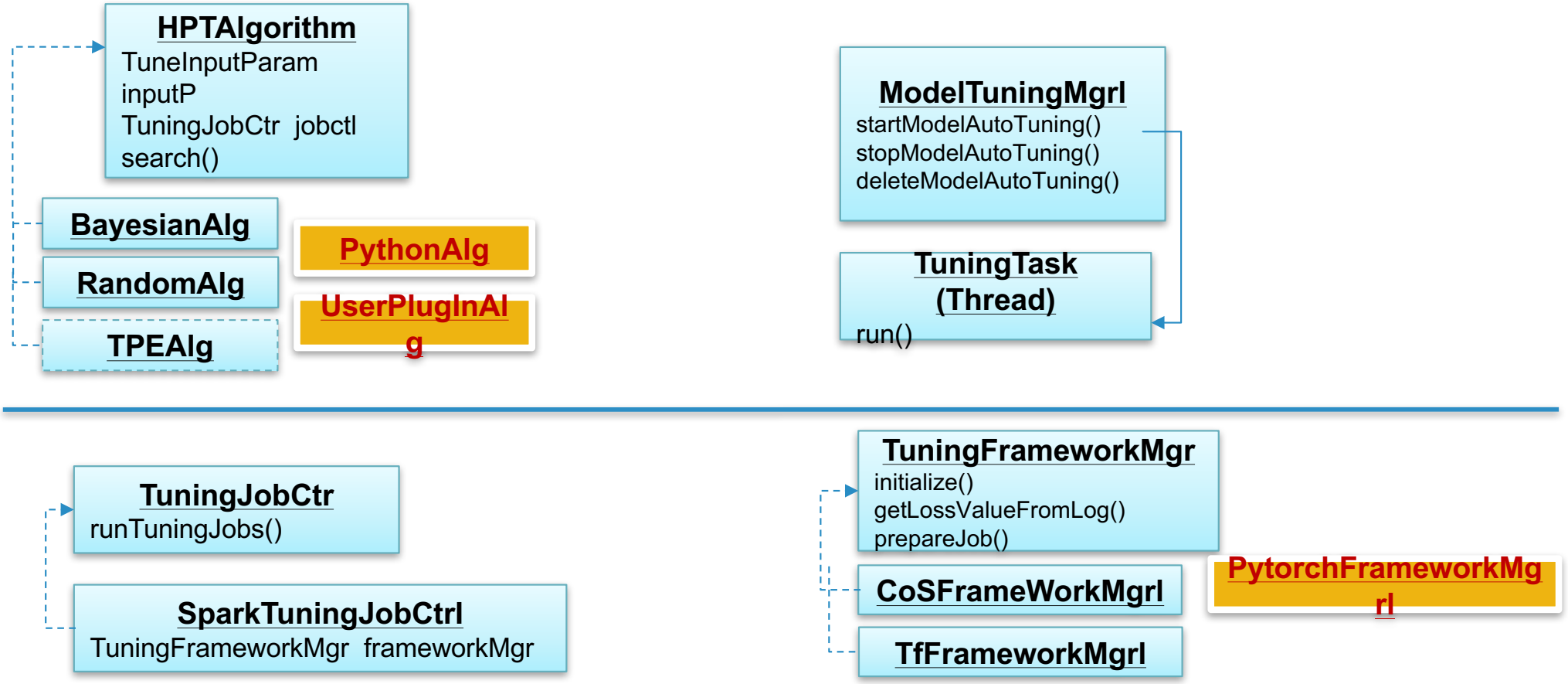
Hyper-Parameter Search Implementation

IBM Spectrum Conductor



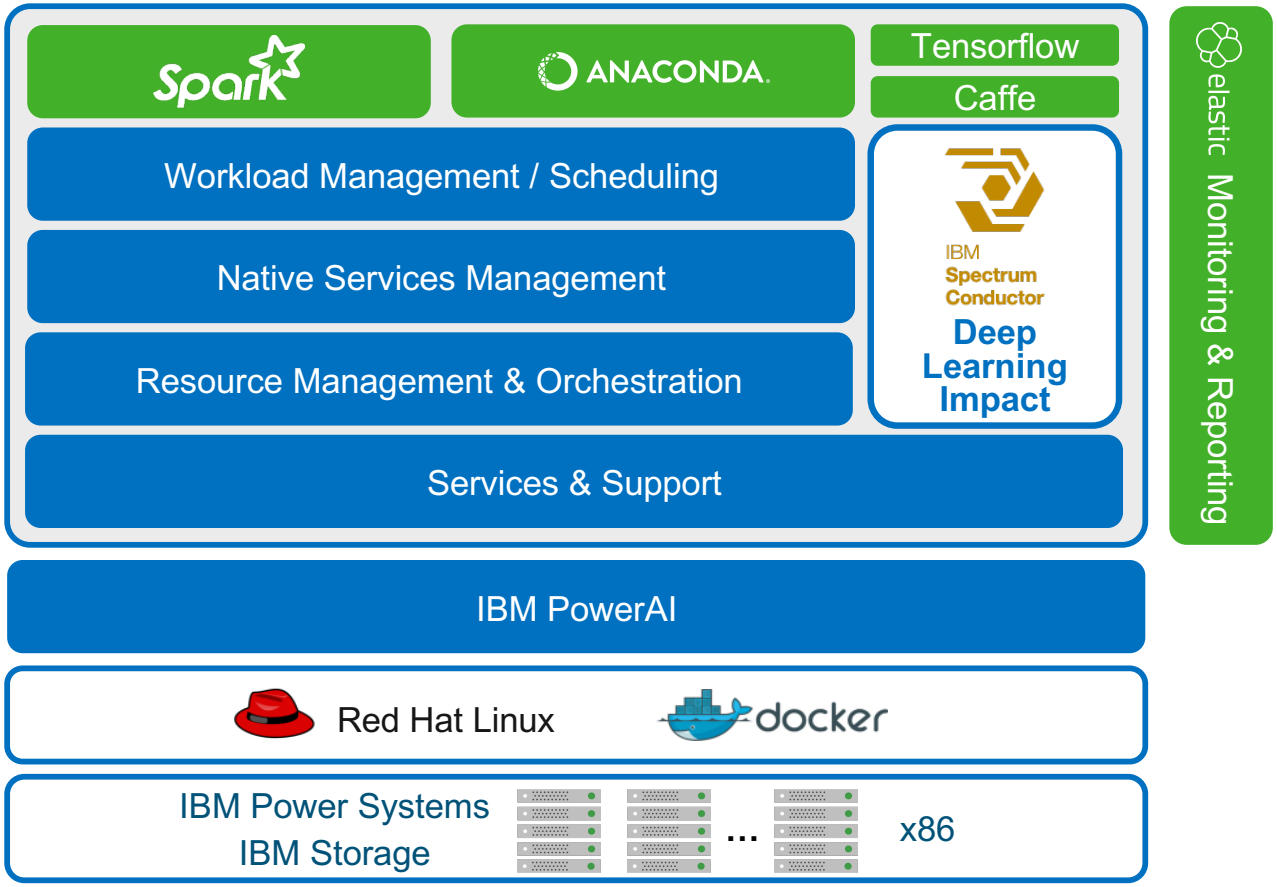
Hyper-Parameter Search Implementation

IBM Spectrum Conductor



Enterprise Class Deep Learning Solution

IBM **Spectrum Conductor Deep Learning Impact**, IBM **PowerAI**, IBM **Storage**



Reference

- [1] David Schaffer, Darrell Whitley and Larry J Eshelman, *Combinations of genetic algorithms and neural networks: A survey of the state of the art*. International Workshop on Combinations of Genetic Algorithms and Neural Networks, 1992.
- [2] J.Snoek, H.Larochelle and R.P.Adams, *Practical Bayesian optimization of machine learning algorithms*. In Advances in Neural Information Processing Systems(NIPS), 2012.
- [3] Bergstra, James and Yoshua Bengio, *Random search for hyper-parameter optimization*. Journal of Machine Learning Research, 2012.
- [4] Lisha Li, Kevin Jamieson and Giulia DeSalvo, *HYPERBAND: BANDIT- BASED CONFIGURATION EVALUATION FOR HYPERPARAMETER OPTIMIZATION*. ICLR, 2017.
- [5] James Bergstra, etc. *Algorithms for Hyper-Parameter Optimization*. Proceedings of the IEEE, 2012.
- [6] Bowen Baker, Otkrist Gupta, Nikhil Naik and RameshRaskar, *DESIGNING NEURAL NETWORK ARCHITECTURES USING REINFORCEMENT LEARNING*. ICLR, 2017.

Please Note

IBM's statements regarding its plans, directions, and intent are subject to change or withdrawal without notice at IBM's sole discretion.

Information regarding potential future products is intended to outline our general product direction and it should not be relied on in making a purchasing decision. The information mentioned regarding potential future products is not a commitment, promise, or legal obligation to deliver any material, code or functionality. Information about potential future products may not be incorporated into any contract.

The development, release, and timing of any future features or functionality described for our products remains at our sole discretion.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve results similar to those stated here.