Neural Network Hyperparameter Optimization

Chris Ouyang (CUHK Mathematics)

Daniel McBride (UTK Mathematics)

2019 JICS RECSEM REU. In collaboration with Kwai Wong (JICS), Stan Tomov (ICL), and Junqi Yin (ORNL)

Presentation Outline

- **Introduction**
- **Learning Curve Matching**
- **Population Based Training with MagmaDNN**

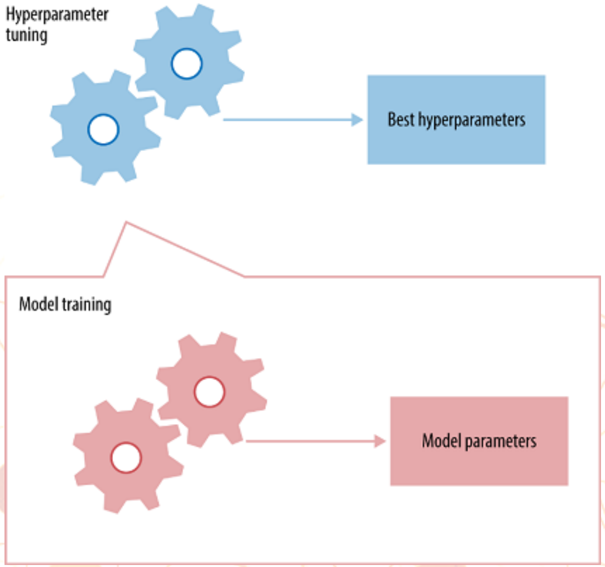
Introduction

- **What is a hyperparameter?**

  They are neural network "presets" like network architecture, learning rate, batch size, and more.

- **Why do we need to optimize the hyperparameters?**

  A poor choice of hyperparameters can cause a network's accuracy to converge slowly or not at all.



towardsdatascience.com

Introduction

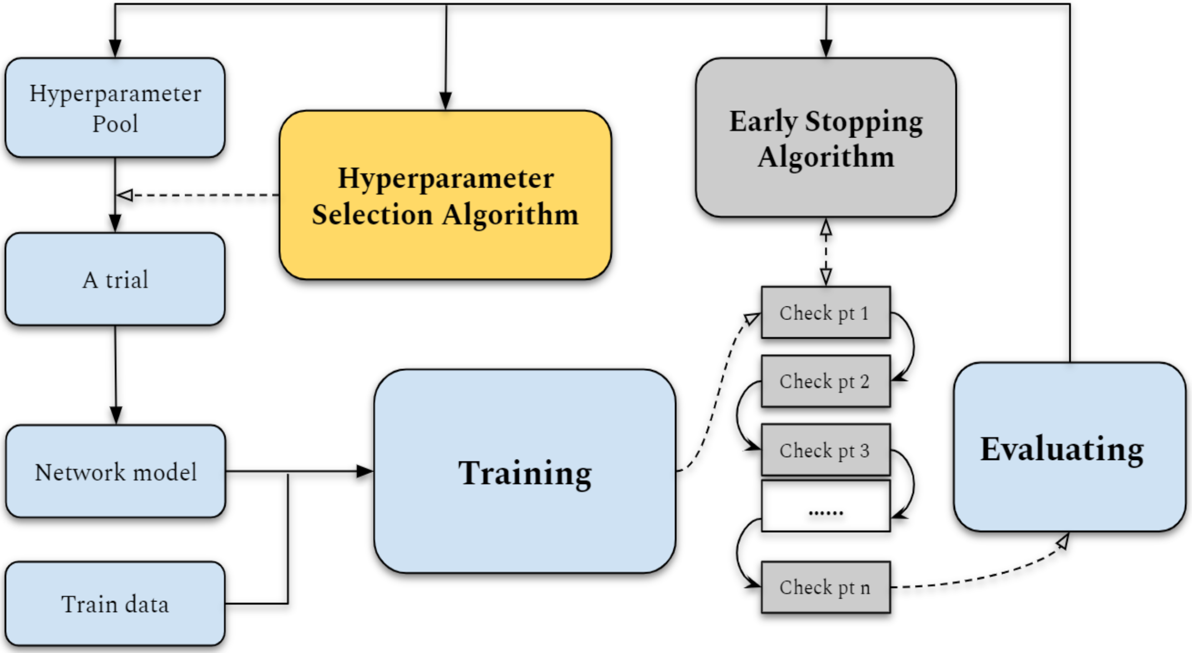
- **What are some obstacles to optimizing hyperparameters?**
  - The Curse of Dimensionality
  - Highly irregular (nonconvex, nondifferentiable) search spaces

- **What are some standard hyperparameter optimization techniques?**
  - Classic Approaches: Grid Search, Random Search
  - Modern Approaches: Early Stopping, Evolutionary Algorithms

An Early Stopping Algorithm Based on Learning Curve Matching

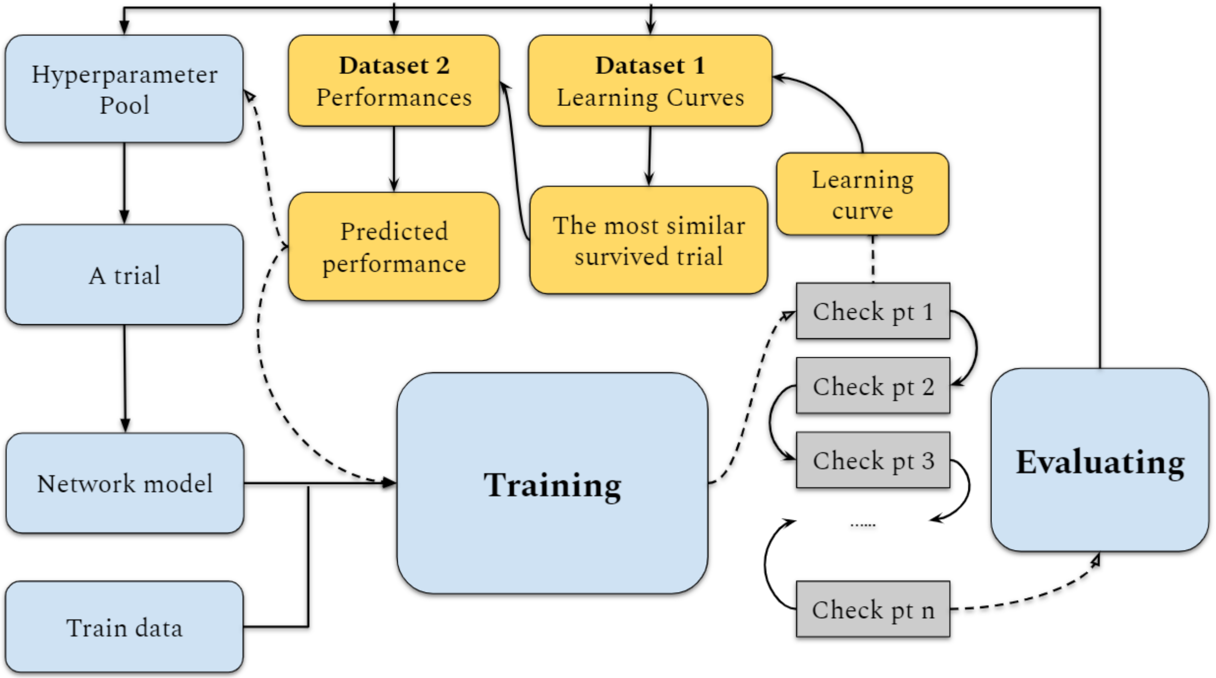
Hyperparameter Algorithms

- **Hyperparameter Selection**: Random search, grid search and Bayesian optimization
- **Early stopping**: Successive Halving Algorithm (SHA) and Hyperband
- **Advanced Algorithm**: Evolutionary Algorithms, such as population based training (PBT) and swarm optimization.

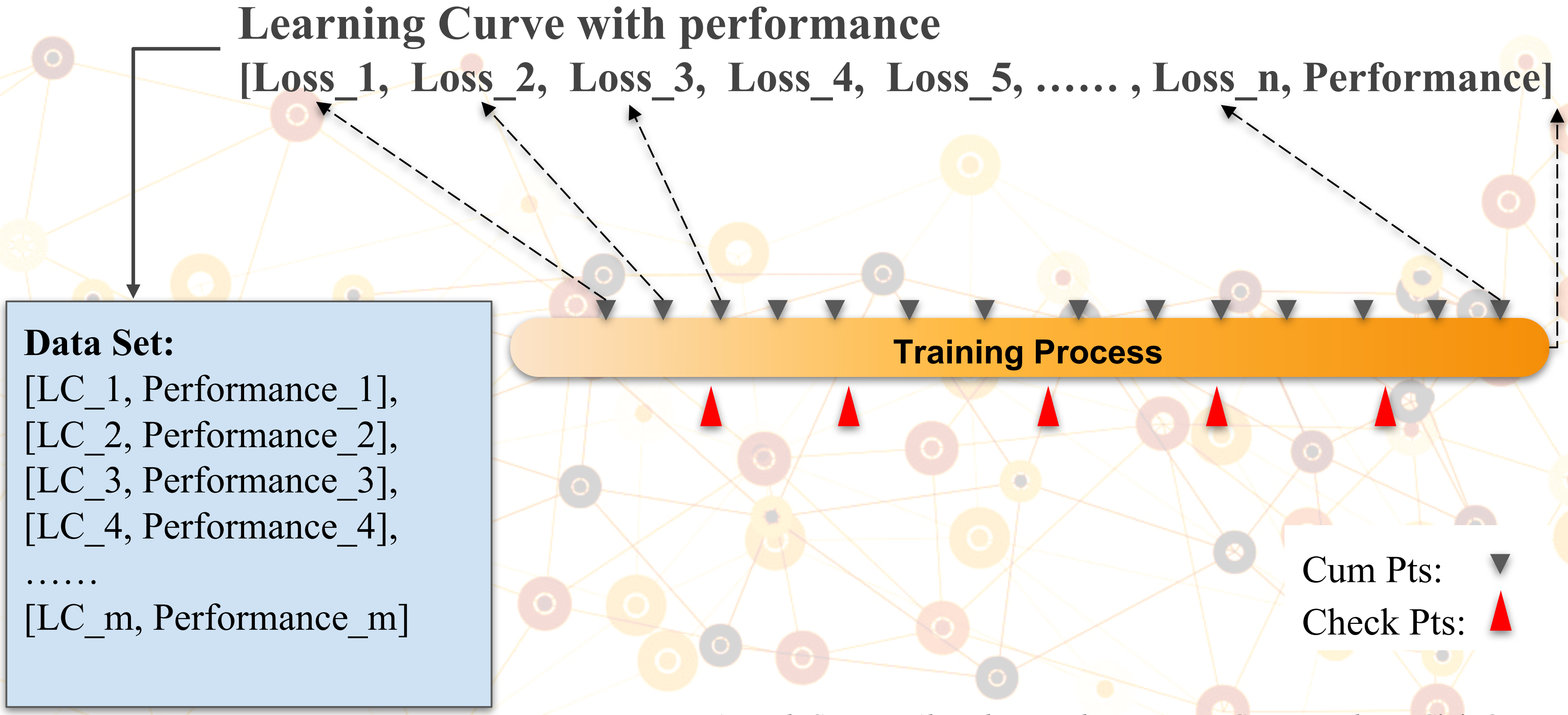


LCM Algorithm: Flow Chart and Terms

- ***Trials***: Sets contain a single sample for every hyperparameter.
- ***Learning Curves***: arrays of the numerical values of the loss function during certain stages of a single training.
- ***Check Points***: points where LCM is applied to decide whether abort the training

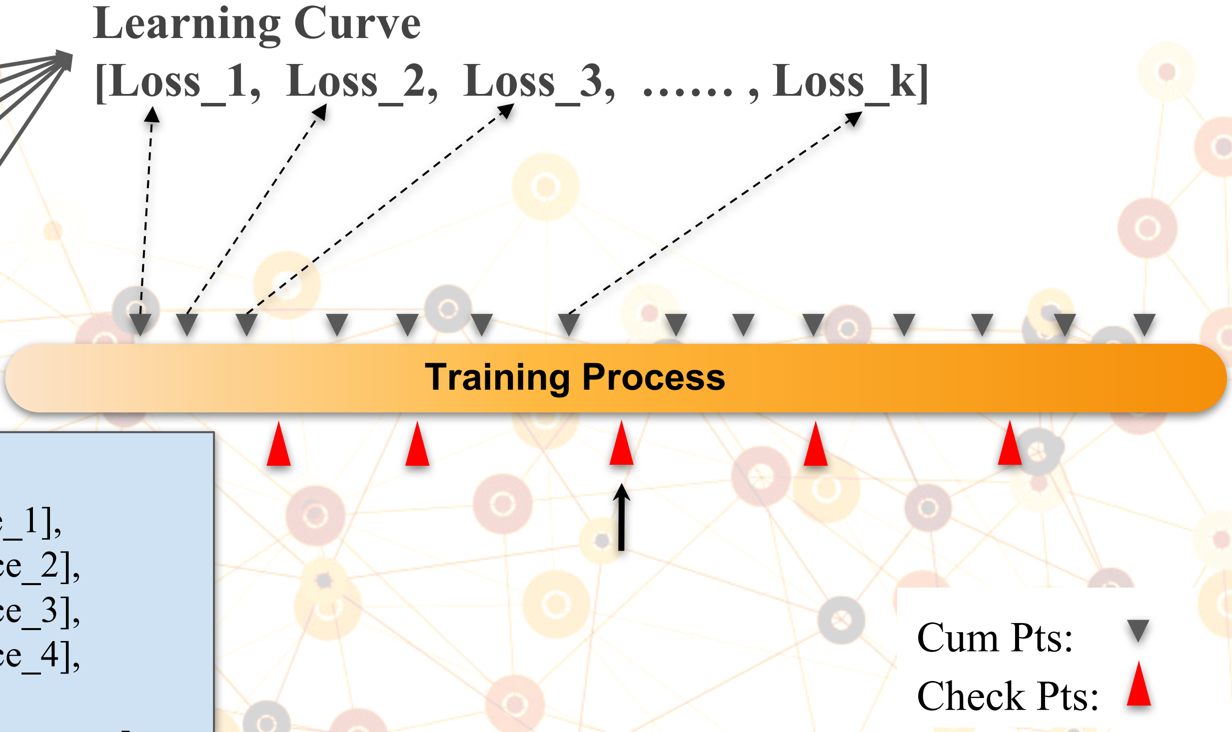


LCM Algorithm: Accumulation Stage



LCM Algorithm: Checking Stage

Data Set:
[L11, L12, L13, ..., L1k],
[L21, L22, L23, … , L2k],
[L31, L32, L33, … , L3k],
[L41, L42, L43, … , L4k],
……
[Lm1, Lm2, Lm3, … , Lmk]



Data Set:
[L11, L12, L13, ..., L1n Performance_1],
[L21, L22, L23, … , L2n Performance_2],
[L31, L32, L33, … , L3n Performance_3],
[L41, L42, L43, … , L4n Performance_4],
……
[Lm1, Lm2, Lm3, … , Lmn, Performance_m]

LCM Algorithm: Pseudocode

Key Steps:

Step 5: Check Trigger for activating checkpoints

Step 6: Starting Training

Step 8-9: Accumulation

Step 10-11: Checking

Algorithm 1 Learning Curve Matching Algorithm

1: **Input:** number of configurations n, early-stopping rate s, split rate r, set of checkpoints C, set of accumulating points A and distance metric d

2: **Initialization:** $T = \texttt{hyperparameter\_configuration\_generator}(n)$, performance list Z = empty list [ ], learning curve list X = empty list [ ]

3: **for** configuration $\theta \in T$ **do**

4:     learning curve γ = empty list [ ]

5:     check trigger $= [\text{length}(Z) > n * r]$

6:     **while** training **do**

7:         training progress $p = \texttt{get\_training\_progress}(\theta)$

8:         **if** $p \in A$ **then**

9:             append$[\gamma, \texttt{get_training_performance}(\theta)]$

10:        **if** $p \in C$ **and** check trigger **then**

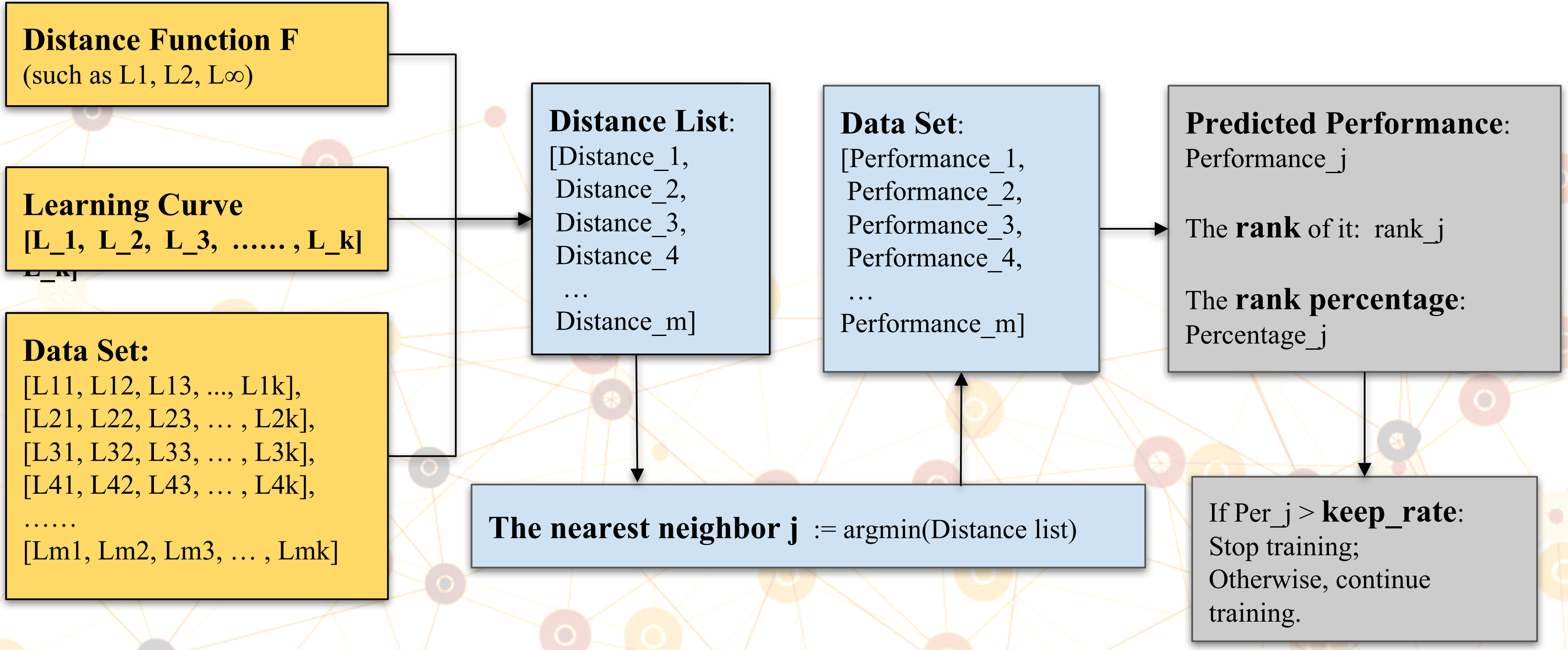
11:            stop_training_trigger $= \texttt{check}(Y, \gamma, d, s)$

12:        append$[X, \gamma]$

13:        append$[Z, \texttt{get_final_performance}(\theta)]$

14: **Output:** the best performance $max(Z)$

LCM Algorithm: Checking Stage



LCM Algorithm: Pseudocode

Y: Dataset; **gamma**: Learning Curve; ***d***: Distance Metric; ***s***: Early-stopping Rate

Algorithm 2 The Function: `check`

function CHECK(Y, γ, d, s)

  $Y = \texttt{fit}(X, \text{length}(\gamma))$ ▷ keep the first length(γ) elements of every previous learning curve for comparison

  $D = \texttt{get\_distances}\ (Y, \gamma, d)$ ▷ compute the distances between γ and every fitted learning curve in Y based on the metric d

  the most similar trial $i = \mathit{argmax}(D)$ ▷ find the most similar trial i

  predicted performance $g = Z[i]$

  rank percentage $q = \texttt{get\_rank\_percentage}(Z, g)$

  Return: $(q > s)$

LCM Algorithm: MNIST Group

- **Network**: Only one dense layer
- **Dataset**: MNIST
- **Optimizer:** Stochastic gradient descent
- **Benchmark**: Random search

Hyperparameter List		
Hyperparameter Name	Data Type	Range
Learning Rate	Float Number	[0, 1]
Momentum	Float Number	[0, 1]
Decay	Float Number	[0, 0.5]
Batch Size	Integer	{32, 64, 96, 144, 192, 288, 376, 512}
Epochs	Integer	{3, 4, 5, 6}

LCM Algorithm: MNIST Group

- Given a fixed number of trials, we compared two algorithms' computing time and best performances. The same experiments are repeated 9 times.

Name	Trials	Computing Time (S)	Best Performance (%)
LCM	100	778.50	97.10
Random	100	3657.75	97.41

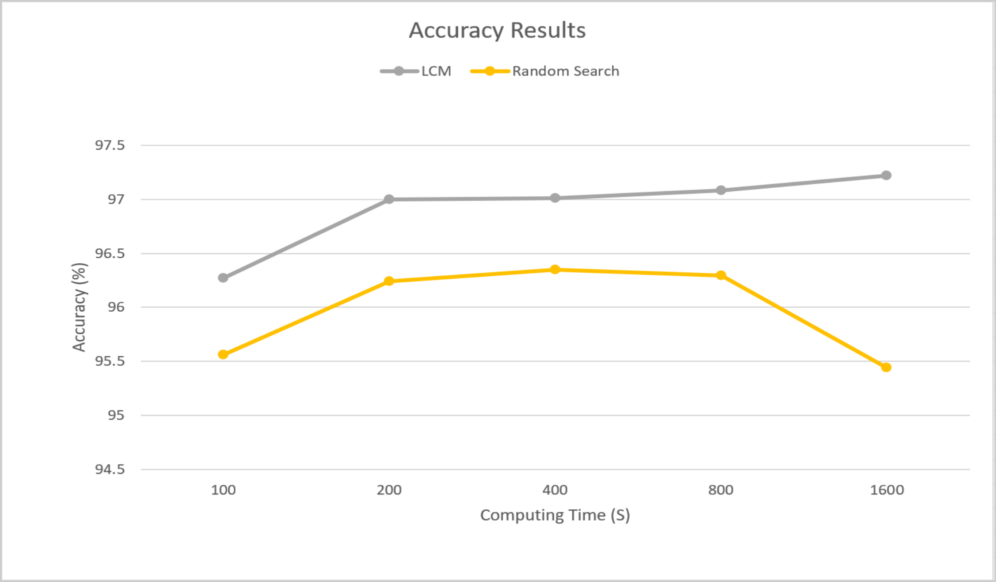
Remark: In 5 of 9 experiments, these two algorithms got the same optimal hyperparameters.

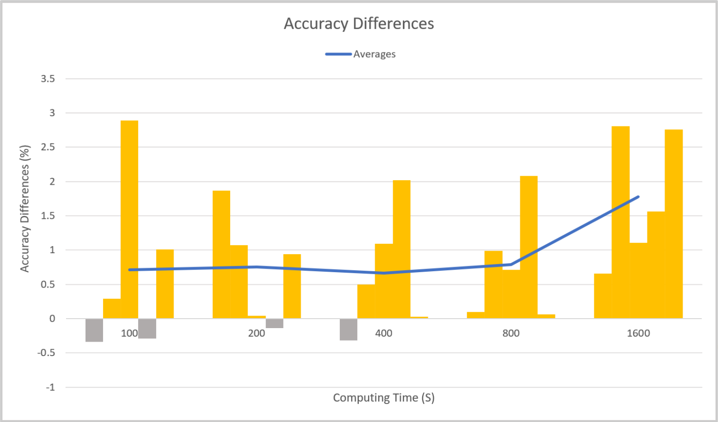
LCM Algorithm: MNIST Group

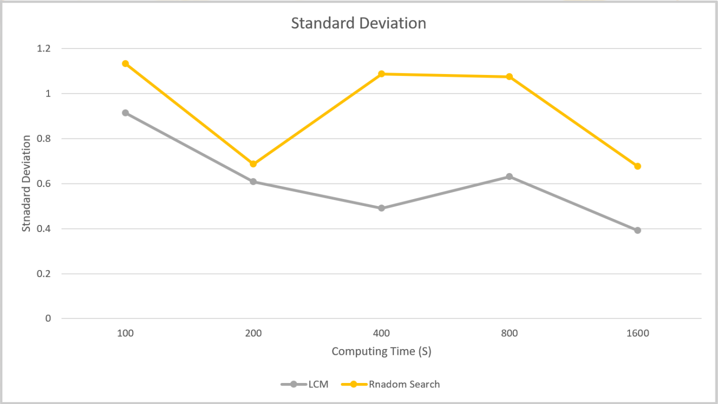
- Given fixed computing time, we compared two algorithms' best performances. The same experiments are repeated 5 times.

Computing time (s)	100	200	400	800	1600
LCM	96.274	96.996	97.01	97.082	97.22
Random	95.562	96.24	96.346	96.294	95.44

LCM Algorithm: MNIST Group







LCM Algorithm: CIFAR10 Group

Hyperparameter List		
Hyperparameter Name	Data Type	Range
Learning Rate	Float Number	[0.001, 0.01]
Beta_1	Float Number	[0.85, 0.95]
Beta_2	Float Number	[0.985, 0.995]
epsilon	Float Number	{1e-07, 1e-06, 1e-08, 5e-07, 5e-06}
Batch Size	Integer	{32, 64, 96, 144, 192, 288, 376, 512}
Epochs	Integer	{10, 15, 20, 25, 30, 35, 40}
Kernel Size of 1st CNN	Integer	{2, 3, 4, 5}
Strides of 1st CNN	Integer	{1, 2}
Dropout After 1st CNN	Float Number	{0.1, 0.2, 0.3, 0.4, 0.5}
Kernel Size of 2nd CNN	Integer	{2, 3, 4, 5}
Strides of 2nd CNN	Integer	{1, 2}
Dropout After 2nd CNN	Float Number	{0.1, 0.2, 0.3, 0.4, 0.5}
Kernel Size of 3rd CNN	Integer	{2, 3, 4}
Strides of 3rd CNN	Integer	{1, 2}
Kernel Size of 4th CNN	Integer	{2, 3, 4}
Strides of 4th CNN	Integer	{1, 2}
Number of Dense Layers After CNN	Integer	{1, 2, 3}
Dropout After Dense	Float Number	{ 0.1, 0.2, 0.3, 0.4, 0.5}

- **Network**: Four CNN layers and several dense layers
- **Dataset**: CIFAR10
- **Optimizer:** Adam
- **Benchmark**: Random search

LCM Algorithm: CIFAR10 Group

- Given a fixed number of trials, we compared two algorithms' computing time and best performances. The same experiments are repeated 12 times.

Name	Trials	Computer Time (S)	Best Performance (%)
LCM	100	8069.08	67.05
Random	100	26498.00	67.26

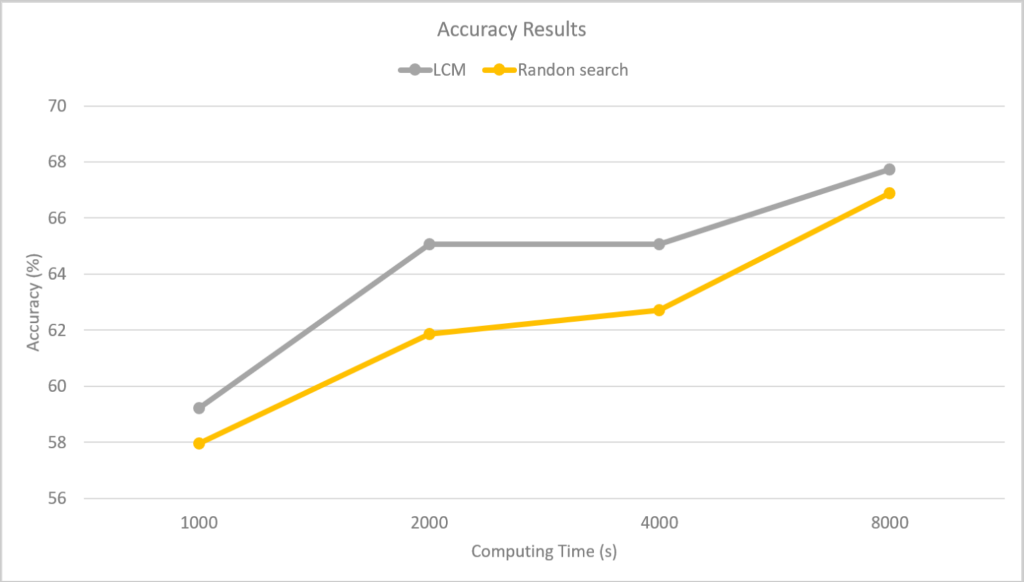
Remark: in 7 of 12 experiments, two algorithms got the same optimal hyperparameters.

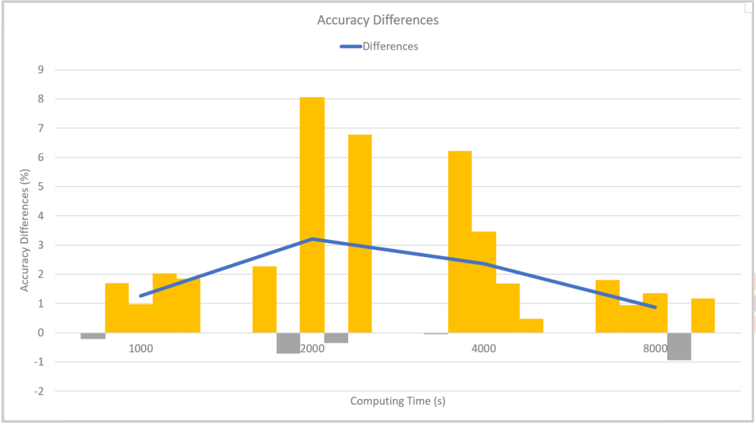
LCM Algorithm: CIFAR10 Group

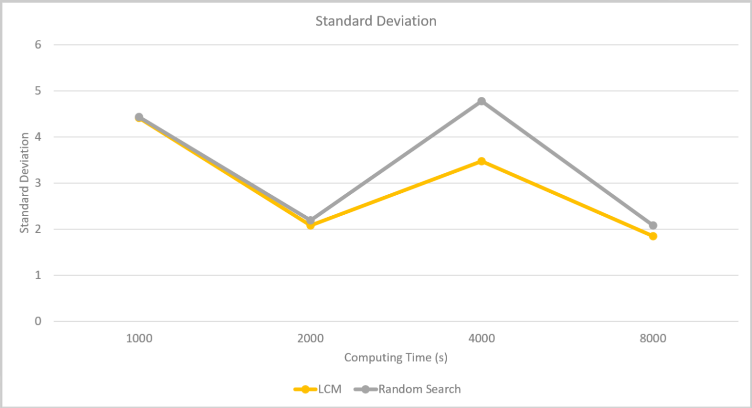
- Given fixed computing time, we compared two algorithms' best performances. The same experiments are repeated 5 times.

Computing time (s)	1000	2000	4000	8000
LCM	59.23	65.06	65.07	67.74
Random	57.96	61.86	62.72	66.88

LCM Algorithm: CIFAR10 Group







LCM Algorithm: Further Discussion

- Neutral Network Design
- Parallel Programming
- New Combinations
- ‘Ultraparameters’

Algorithm 3 Asynchronous Learning Curve Matching

Input: number of configurations n, early-stopping rate s, split rate r, set of checkpoints C, set of accumulating points A and distance metric d

Initialization: T = `hyperparameter_configuration_generator`(n), performance list Z = empty list [ ], learning curve list X = empty list [ ]

while free worker **do**

    θ = `get_new_one`(T) ▷ Return a new configuration for training.

    check trigger = `get_check_trigger()`

    for every check point $p \in C$ **do**

        learning curve γ = `update_lc`(θ, p) ▷ Update the learning curve until meeting the checkpoint p.

        `send_to_supervisor`(γ)

        stop_training_trigger = `receive_from_supervisor()`

    z = `get_final_performance`(θ)

    `send_to_supervisor`(γ, z)

for supervisor worker **do**

    for γ = `receive_from_worker()` **do**

        trigger = = `check`(Y, γ, d, s) ▷ This function refers to Algorithm 2.

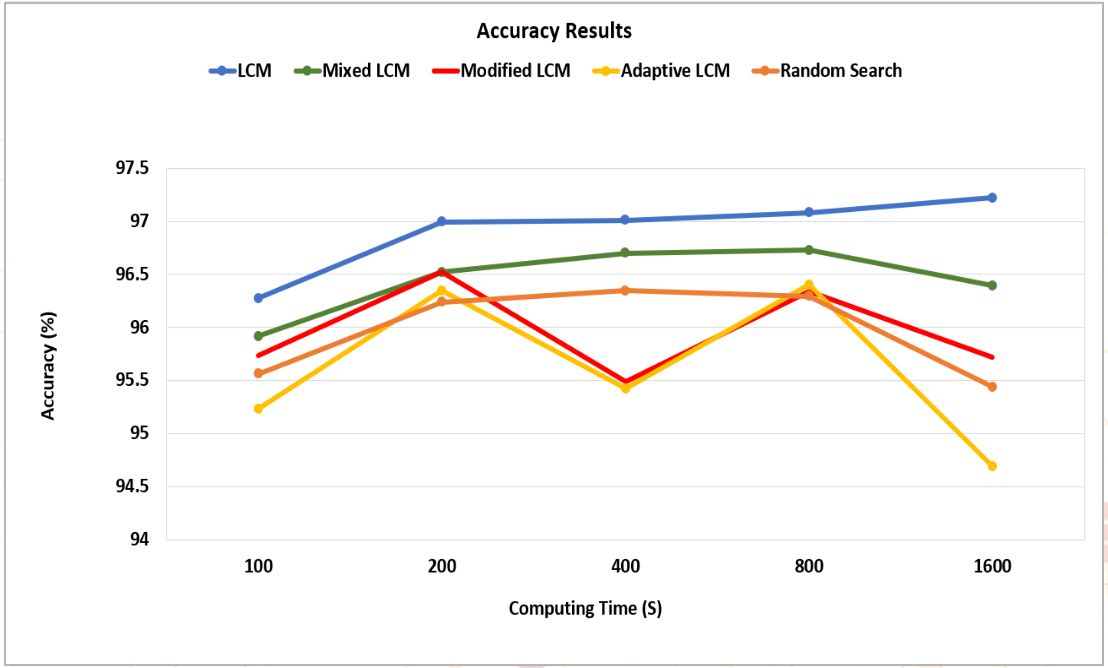
        `send_to_worker`(trigger)

    for γ, z = `receive_from_worker()` **do**

        X, Z = `update`(X, Z, γ, z)

        check trigger = [length$(Z) > n * r$]

LCM Algorithm: Other Work



Population Based Training with MagmaDNN

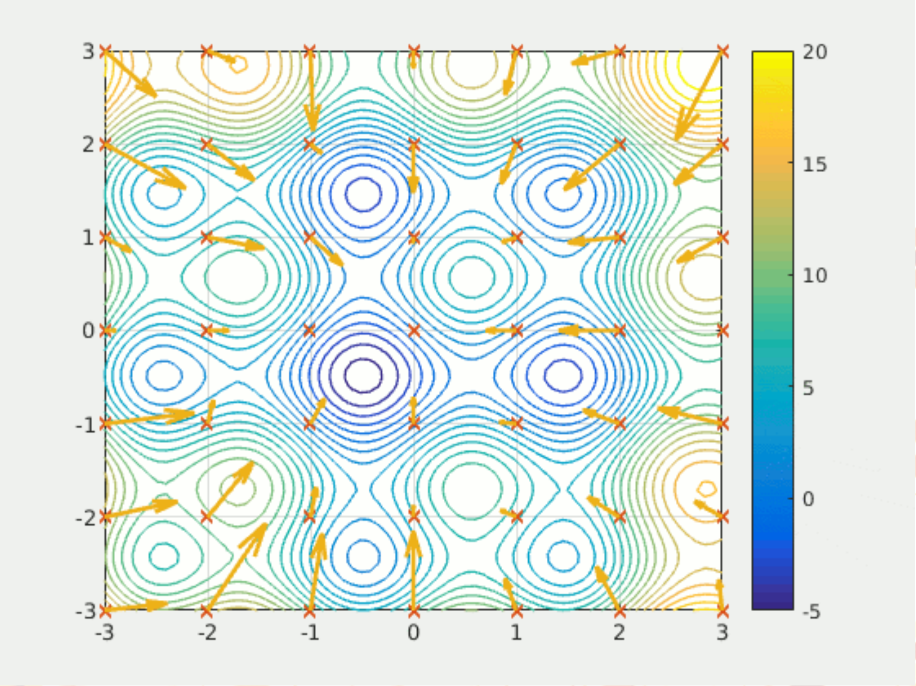
PBT: Background

- **What is Population Based Training (PBT)?**

  PBT is an evolutionary hyperparameter optimization algorithm.

- **Evolutionary optimization algorithm**
  use natural models to inspire a particular approach to traversing a search space to find the minimum o an objective function. One classic case is the Particle Swarm Optimization algorithm, inspired b the swarming behavior of bees.

Particle Swarm Optimization



wikimedia

PBT: Background

- **What are the benefits of PBT?**

  PBT outperforms the standard hyperparameter tuning benchmarks. These benchmark algorithms, **Grid Search and Random Search**, each have their own limitations, which PBT overcomes.

- **Why should we implement it on MagmaDNN?**
  - MagmaDNN is engineered for high performance computing on large distributed systems.
  - The current standard implementation (Ray-Tune: shared memory model) has a scalability bottleneck.

PBT: Algorithm

How does the PBT Algorithm work?

- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling

PBT: Algorithm

How does the PBT Algorithm work?

- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling

Step 1

Initialize Networks
Random Weights
Random Hyperparameters

Image from arcfertility.com

PBT: Algorithm

How does the PBT Algorithm work?

- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling

Step 2

Training Period

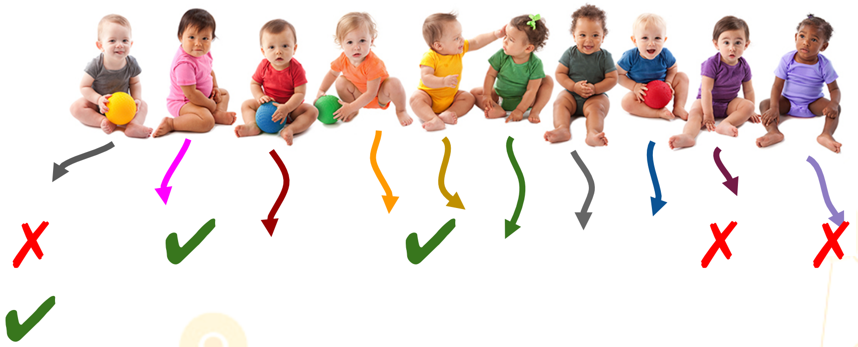
Networks optimize weights in the usual way (SGD, ADAM, etc.)

Image from arcfertility.com

PBT: Algorithm

How does the PBT Algorithm work?

- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling



Step 3

Rank Fitness

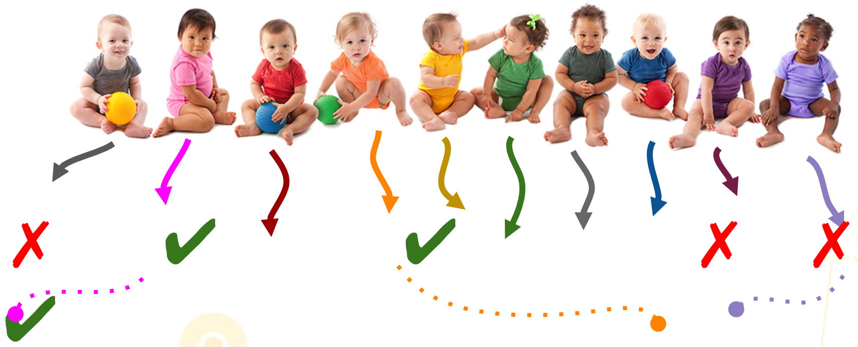
accuracy, loss or other measure determines most and least fit

Image from arcfertility.com

PBT: Algorithm

How does the PBT Algorithm work?

- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling



Step 4

Exploit

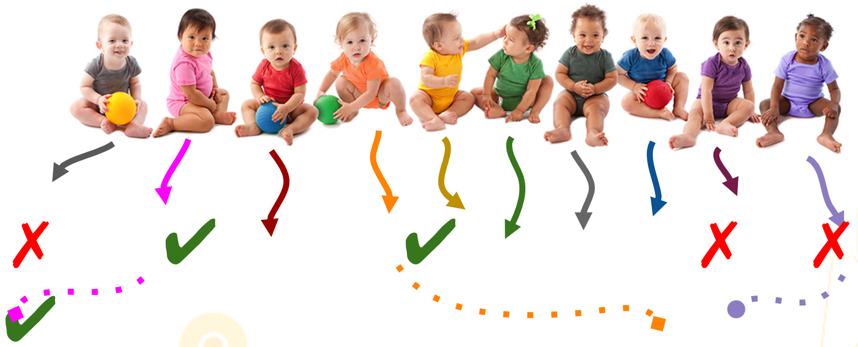
Copy the weights and hyperparameters from the most fit to the least

Image from arcfertility.com

PBT: Algorithm

How does the PBT Algorithm work?

- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling



Step 5

Explore

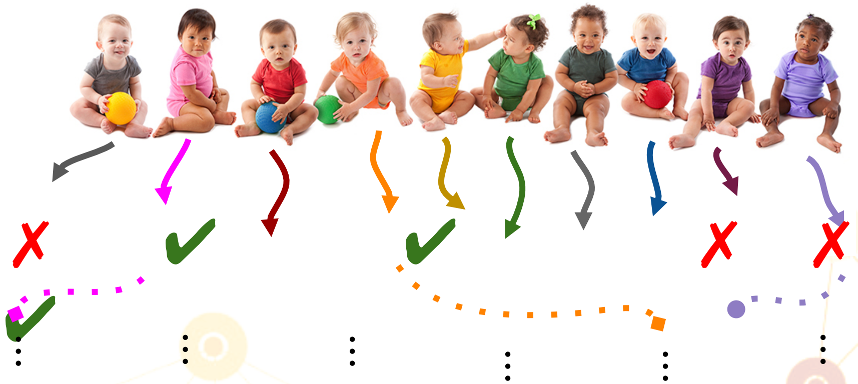
Perturb the updated hyperparameters

Image from arcfertility.com

PBT: Algorithm

How does the PBT Algorithm work?

- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling



Repeat

Train -> Exploit -> Explore
process until desired convergence

Image from arcfertility.com

PBT: Algorithm

How does the PBT Algorithm work?

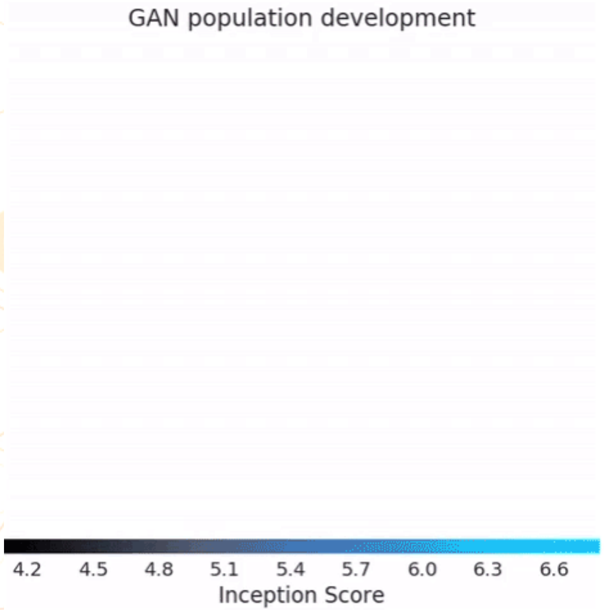
- Population Model
- Stochasticity
- Exploit / Explore
- Early Stopping
- Evolution
- Adaptive Hyperparameter Scheduling

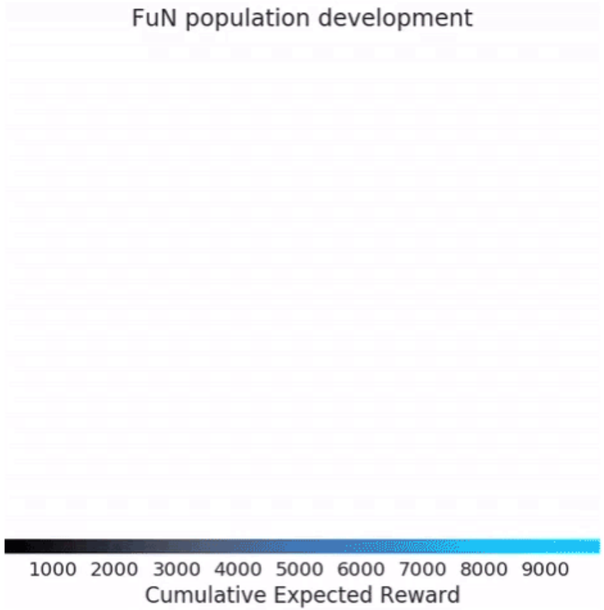
End Result: Optimized networks with optimized hyperparameter schedules

Images from arcfertility.com and csworldaffairs.org

PBT: Algorithm

How does the PBT Algorithm work?





Jaderberg et al.

PBT: Algorithm

Does PBT's functionality improve on the benchmark algorithms?

	Grid Search	Random Search	PBT
Parallelizability	✔	✔	✔
Stochasticity	✗	✔	✔
Early Stopping	✗	✗	✔
Adaptive Hyperparameters	✗	✗	✔

PBT: Analysis - Learning Rate Optimization

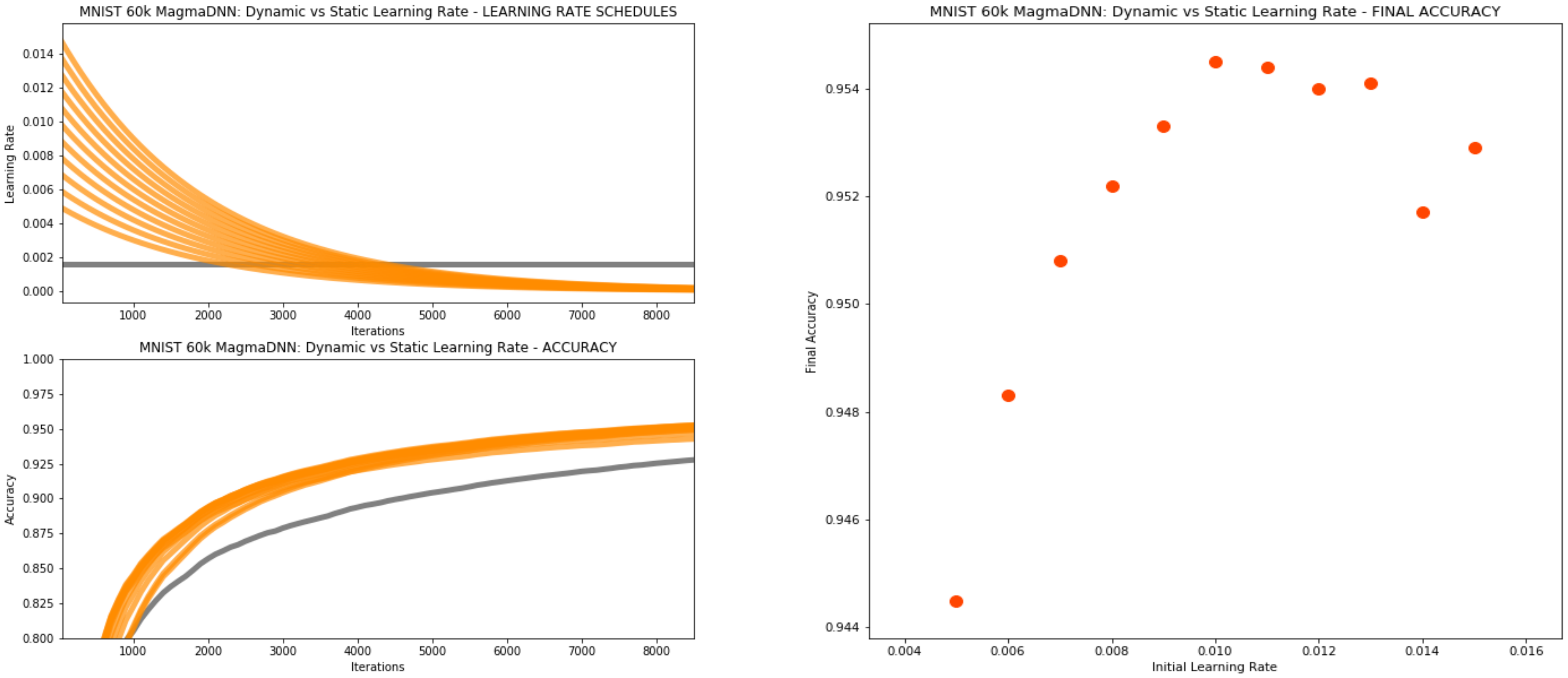
- **Data: MNIST**
  - **60k images of handwritten digits 0-9**
  - **256 greyscale pixels per image**
  - **10 categories (0-9)**

- **Network Backend: MagmaDNN**
  - **Network Structure: In -> FCB -> Sig -> FCB -> Sig -> FCB -> Out**
  - **Weight Optimizer: Stochastic Gradient Descent**
  - **Number of Epochs = 5**
  - **Batch Size = 32**

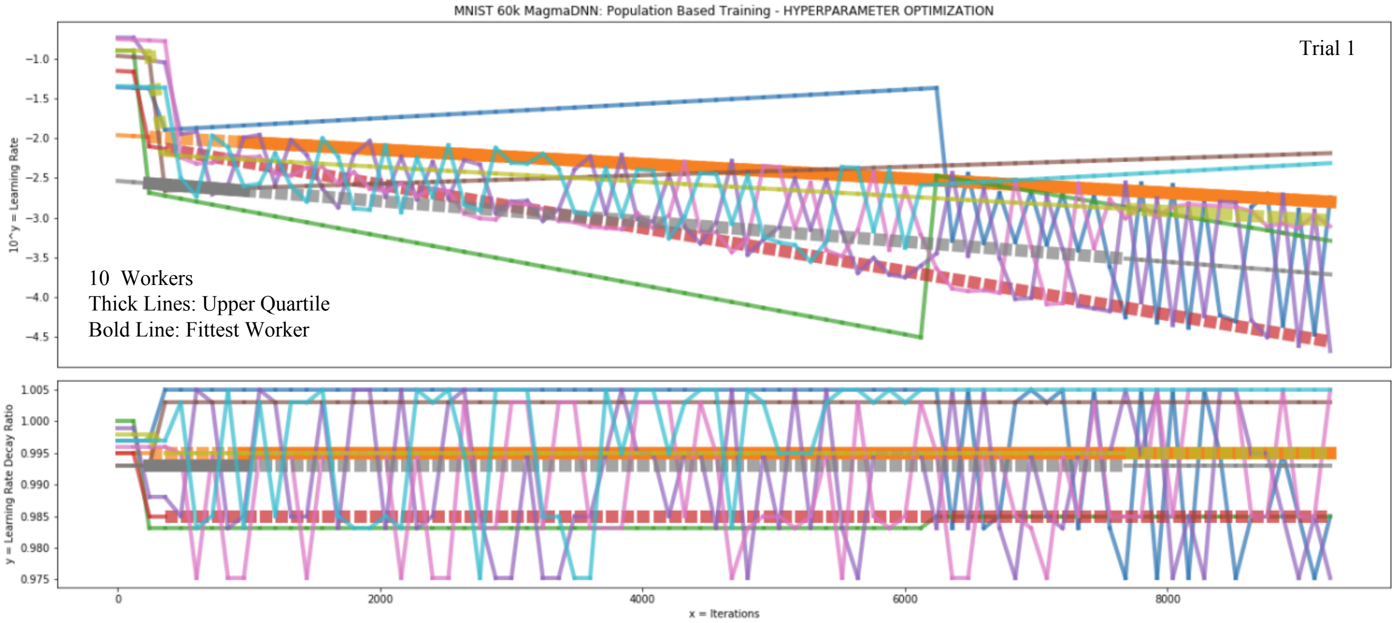
- **Communication Backend: MPI**

*FCB := Fully Connected Layer with Bias
*Sig := Sigmoid

PBT: Analysis - Learning Rate Optimization



PBT: Analysis - Learning Rate Optimization

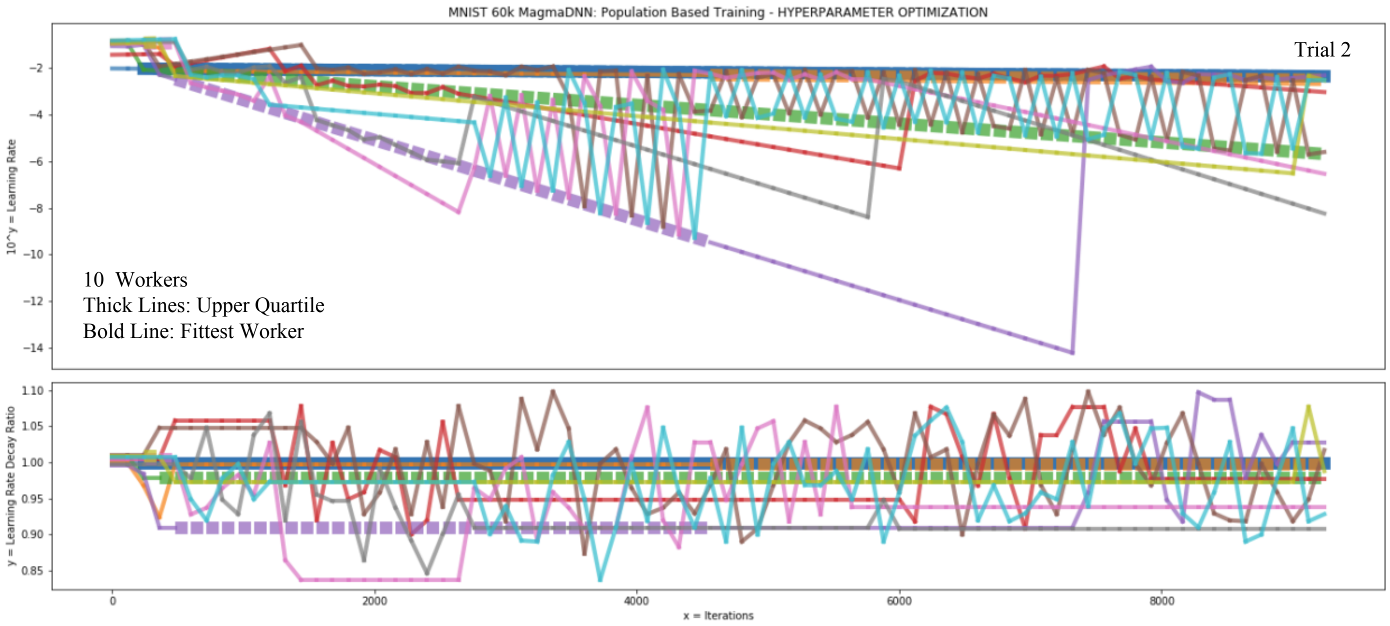


Worker	Trial 1 Final Accuracy
1	0.9251
2	0.9574
3	0.9281
4	0.9468
5	0.925
6	0.9258
7	0.9258
8	0.9378
9	0.9411
10	0.9278

Trial 1 Specifications

LR Sampling Distribution	Uniform Random between .0001 and .2
LR Decay Ratio Sampling Distribution	Uniform Random between .99 and 1
LR Decay Pace	Every 20 iterations
Evolution Pace	Every 120 iterations
LR Perturbation Distribution	120% and 80% equally likely
LR Decay Ratio Perturbation Distribution	99% and 101% equally likely

PBT: Analysis - Learning Rate Optimization



Worker	Trial 2 Final Accuracy
1	0.957
2	0.9322
3	0.9419
4	0.9155
5	0.9151
6	0.9112
7	0.9161
8	0.916
9	0.9146
10	0.9114

Trial 2 Specifications

LR Sampling Distribution	Uniform Random between .0001 and .2
LR Decay Ratio Sampling Distribution	Uniform Random between .98 and 1.01
LR Decay Pace	Every 20 iterations
Evolution Pace	Every 120 iterations
LR Perturbation Distribution	No perturbation
LR Decay Ratio Perturbation Distribution	Uniform Random between 90% and 110%

Conclusions

- Dynamic and adaptive learning rate optimization, such as that deployed in our MagmaDNN PBT implementation, improves the convergence of neural networks.
- Early stopping hyperparameter tuning algorithms, such as LCM, can compete with standard benchmarks like Random Search.

Future Work

- Program more custom MagmaDNN classes to explore the tuning of Convolutional Neural Network hyperparameters.
- Implement LCM using the MagmaDNN framework.
- Complete an implementation of MagmaDNN PBT utilizing OpenDIEL's distributed workflow system.

Thanks for listening!

-The Hyperparameter Team

Acknowledgements: National Science Foundation, Joint Institute of Computational Sciences (UT-ORNL), Extreme Science and Engineering Discovery Environment (XSEDE), BP HPC Team.

References: Bergstra and Bengio, *Random Search for Hyperparameter Optimization*, 2012; Goodfellow et al, *Deep Learning*, 2016; Jaderberg et al, *Population Based Training of Neural Networks*, 2017.