Neural Network Hyperparameter Optimization with MagmaDNN and OpenDIEL

The University of Tennessee

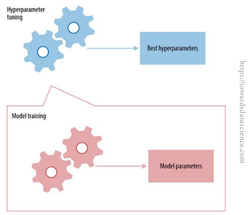
The Chinese University of Hong Kong

Students: Qiqi Ouyang (CUHK), Daniel McBride (UTK), Rocco Febbo (UTK)
Mentors: Kwai Wong (UTK), Stan Tomov (UTK), Junqi Yin (ORNL)

Joint Institute for Computational Sciences

National Science Foundation

• Introduction •



What is a hyperparameter? Hyperparameters 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.
What are some obstacles to optimizing hyperparameters? The Curse of Dimensionality: the search space grows exponentially with each new hyperparameter. Also, highly irregular (nonconvex, nondifferentiable) search spaces are the norm.
What are some standard hyperparameter optimization techniques? Classic approaches include *Grid Search* and *Random Search*, while more modern approaches are *Early Stopping*, *Evolutionary Algorithms*, and *Dynamic Learning Rate.*

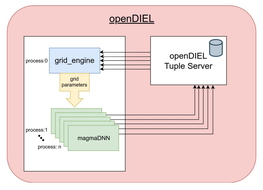
• Research Objectives •

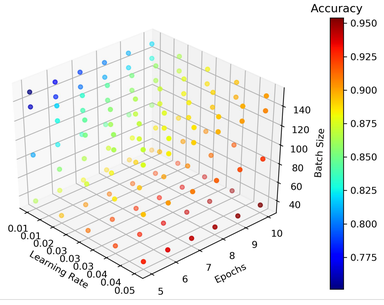
- Extend the HPC machine learning framework, MagmaDNN, to include more hyperparameter tuning, including evolutionary functionality.
- Improve the OpenDIEL Grid Search hyperparameter tuning application.
- Implement a novel hyperparameter tuning algorithm amenable to use on supercomputing scale using modern optimization techniques.
- Explore the performance of this new algorithm on various network architectures, comparing it to classic benchmarks like random search.
- Experimentally demonstrate the benefits of dynamic learning rate over static learning rate.

• OpenDIEL Grid Search •

Since hyperparameter tuning is so computationally intensive it is desirable to have a distributed system which manages the process. Thankfully, the process is inherently parallelizable due to the small amount of data required to do a very large amount of work.

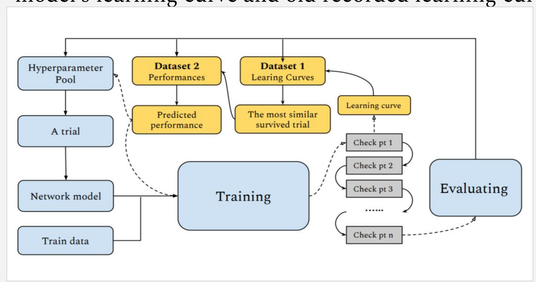
The flow chart on the upper right shows how the grid engine is implemented in **OpenDIEL**. One single process manages sending the hyperparameters to the other processes. They train, then send their metrics back to process 0. The graph below it shows the output of the system when training across a 3D grid.





• Learning Curve Matching •

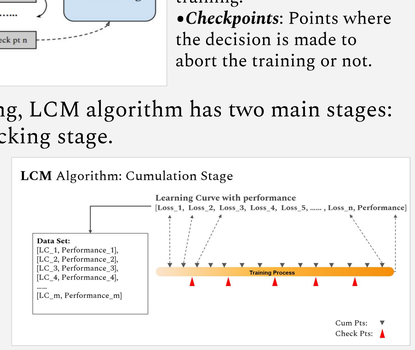
The flow chart below indicates the process of the early stopping algorithm, **Learning Curve Matching** (**LCM**). After setting the hyperparameters, the model will go through every checkpoint, where the stopping action will be triggered based on the comparison between the model's learning curve and old recorded learning curves.



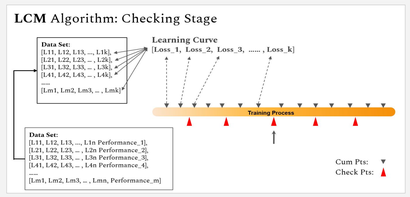
- ***Trials***: Sets that contain a single sample for every hyperparameter.
- ***Learning Curves***: Arrays of the numerical values of the loss function at certain stages during a single training.
- ***Checkpoints***: Points where the decision is made to abort the training or not.

During the hyperparameter tuning, LCM algorithm has two main stages: the accumulation stage and the checking stage.

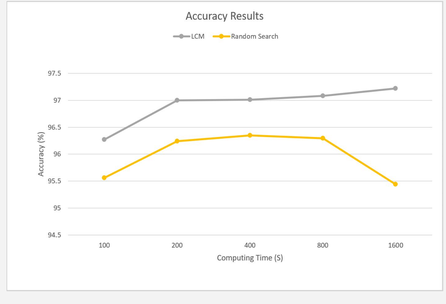
- In the accumulation stage, only cumulative points are activated. The value of the loss function at cumulative points and the corresponding final performances during every complete training will be collected for the next stage.

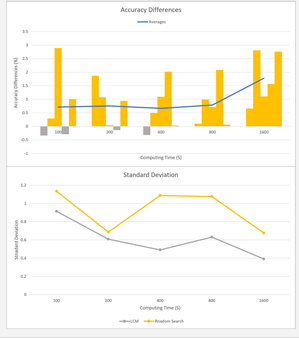


- In the checking stage, both the cumulative points and checkpoints will be activated. Besides collecting needed data as in the accumulation stage, curve comparison and early stopping will also be implemented.



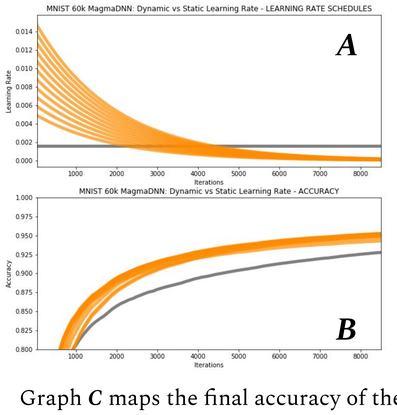
At every checkpoint the Euclidean distances between the new learning curve and the old learning curves will be calculated. Then the most similar complete training will be identified based on these distances, and its performance will be viewed as the predicted performance of the model in training. The rank percentage of the predicted performance will decide whether to early stop the training. The result of experiments based on MNIST is shown below. LCM gets a slightly better and more stable performance than random search.



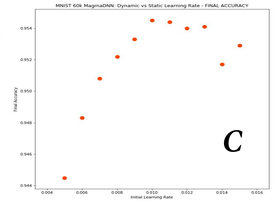


• Dynamic Learning Rate •

The following graphs show the results obtained from an experiment comparing static to dynamic learning rate. Using **MagmaDNN**, a network with three fully connected layers was trained on MNIST with stochastic gradient descent. Batch size was constant across trials, at 32 per iteration.



Graph ***A*** plots learning rate against number of training iterations. It shows a gray line, a trial with static learning rate of ~0.0016, while the orange lines plot trials with decaying learning rate with variable initial values. The decay rate was 5% every 100 iterations.
Graph ***B*** shows that all of the dynamic trials achieved greater accuracy more quickly. The Y-axis is accuracy, while the X-axis is number of training iterations. These plots affirm, in our case, the benefit of dynamic learning rate.



Graph ***C*** maps the final accuracy of the dynamic trials against their initial learning rate values. The greatest final accuracy is steadily approached from below, as initial learning rate increases. The best final accuracy is achieved with an initial learning rate value of 0.01, and then for greater initial learning rate values, the final accuracy exhibits irregular behavior.
These results confirm that lower learning rates, while slowing convergence, contribute to stability.

Population Based Training of Gradien Descent Learning Rates
The plot below shows the evolution of learning rate schedules of a population of **MagmaDNN** neural networks training in parallel on the MNIST dataset, with above specifications. The algorithm is implemented with MPI, with distributed, supercomputer scale application in mind. Each network dynamically tracks its fitness rank among the population. The least fit quartile is replaced by evolved copies of the most fit quartile, their learning rates adaptively updated via perturbation of the parent values.

• Future Work •

- Implement Population Based Training using OpenDIEL.
- Extend MagmaDNN functionality to include hyperparameter tuning with Learning Curve Matching.

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.