Optimizing Deep Learning Hyper-Parameters Through an Evolutionary Algorithm

Steven R. Young, Derek C. Rose, Thomas P. Karnowski, Seung-Hwan Lim, Robert M. Patton
Oak Ridge National Laboratory
PO Box 2008, MS-6085
Oak Ridge, TN 37831
youngsr@ornl.gov

ABSTRACT

There has been a recent surge of success in utilizing Deep Learning (DL) in imaging and speech applications for its relatively automatic feature generation and, in particular for convolutional neural networks (CNNs), high accuracy classification abilities. While these models learn their parameters through data-driven methods, model selection (as architecture construction) through hyper-parameter choices remains a tedious and highly intuition driven task. To address this, *Multi-node Evolutionary Neural Networks for Deep Learning (MENNDL)* is proposed as a method for automating network selection on computational clusters through hyper-parameter optimization performed via genetic algorithms.

Categories and Subject Descriptors

I.2.6 [**Computing Methodologies**]: Artificial Intelligence – *Learning.*
I.2.8 [**Computing Methodologies**]: Artificial Intelligence – *Problem Solving, Control Methods, and Search*.

General Terms

Algorithms, Performance, Theory.

Keywords

Deep Learning, Evolutionary Algorithm, Hyper-Parameter Optimization, Convolutional Neural Networks.

1. INTRODUCTION

Deep Learning is a sub-field of machine learning that focuses on learning features from data through multiple layers of abstraction. These features are learned with little human domain knowledge and have dramatically improved state of the art in many applications [1, 9, 10]. Due to the inability for any one network to best generalize for all datasets, a necessary step before applying DL to a new dataset is selecting an appropriate set of hyper-parameters. For CNNs, these hyper-parameters include the number of layers, the number of hidden units per layer, the activation function for a layer, the kernel size for a layer, the arrangement of these layers within the network, etc. Selecting a new network topology for a previously unused dataset can be time-consuming and tedious task. The number of hyper-parameters being tuned and the evaluation time for each new set of hyper-parameters makes their optimization in the context of deep learning particularly difficult.

Studies of the effects of hyper-parameters on different deep learning architectures have shown complex relationships, where hyper-parameters that give great performance improvements in simple networks do not have the same effect in more complex architectures [5]. These studies also show that the results on one dataset may not transfer to another dataset with different image properties, prior probability distributions, number of classes, or number of training examples. With no explicit formula for choosing a correct set of hyper-parameters, their selection often depends on a combination of previous experience and trial-and-error. Due to the computationally expensive nature of DL algorithms, which can take days to train on conventional platforms, repeated trial and error is both inefficient and not thorough. While the DL community has settled on a set of hyper-parameters that often work well for common image and speech recognition problems, it's not clear that such parameters extend to domains with different data modality or structure or if such hyper-parameters are indeed optimal. This work proposes to address the model selection problem and ease the demands on data researchers using MENNDL, an evolutionary algorithm that leverages a large number of compute nodes. These nodes communicate over MPI to distribute the task of finding the optimal hyper-parameters across the nodes of a super computer. Our software makes use of the Convolutional Architecture for Fast Feature Embedding (Caffe) package [11] for implementing the DL networks.

2. RELATED WORK

The three most widely used methods for hyper parameter selection in deep learning are (1) manual search, (2) grid search, and (3) random search [2]. Manual search refers to the process of a

MLHPC2015, November 15-20, 2015, Austin, TX, USA
© 2015 ACM. ISBN 978-1-4503-4006-9/15/11…$15.00
DOI: http://dx.doi.org/10.1145/2834892.2834896

researcher manually selecting hyper-parameter sets to evaluate. This is often chosen for its ease and can reach a reasonable solution quickly as an intuition for the importance of the various hyper-parameters is developed. However, it is difficult to reproduce results on new data using this method, particularly for non-experts. Grid search allows for reproducible results, but is not efficient in searching a high-dimensional hyper-parameter space. It is widely used since it is quick to implement, trivial to parallelize, and intuitively would allow the entire search space to be explored. However, grid search wastes resources exploring what could be unimportant dimensions of the hyper-parameter space while holding all other values constant, some of which could be more important. Thus, random search is much more effective at searching the hyper-parameter space since it avoids this problem of under-sampling important dimensions, all while being just as easy to implement and just as easy to parallelize as grid search [2]. Random search suffers from being non-adaptive (i.e. the hyper-parameter sets selected to be evaluated are not selected using results that are already available) and can be outperformed by a combination of manual and grid search on some problems [2]. [3, 18] have applied Bayesian optimization techniques for designing convolutional vision architectures by learning a probabilistic model over the hyper-parameter search space. These papers have produced architectures that surpassed the performance of those built by domain experts, with [18] showing a 3% gain in accuracy for the CIFAR-10 dataset with a Gaussian process method. [4] compared similar hyper-parameter optimization algorithms to determine their efficiency in selecting parameters for candidate problems with known solutions.

Genetic algorithms provide the opportunity to both search the hyper-parameter space in a random fashion and also utilize previous results to direct the search. Prior work in using genetic algorithms (GAs) with neural networks falls into two categories. The first focuses on replacing stochastic gradient descent and backpropagation with a GA [12, 13, 15, 16, 20, 21]. While these works provide an interesting alternative training method, the approach is quite different than hyper-parameter optimization that is the focus of this work. The second category consists of works where a genetic algorithm is applied to optimize the hyper-parameters of shallow networks that don't share the large hyper-parameter space seen in deep networks [6, 8]. [6] performed a comprehensive empirical evaluation of existing combinations of EAs with shallow neural networks (including hybridizing network parameter GA search and local optimizers). Their systematic evaluation on a large variety of datasets focused only on small networks (31 hidden units and single hidden layer), so it is difficult to draw conclusions for more demanding representation problems [8] focuses on training a GA to define the connectivity within a shallow neural network with network performance as the fitness for each population member. The resulting network outperforms both a fully-connected network and the best randomly connected network from the first generation. The effectiveness of these methods in finding improved sets of hyper-parameters on smaller neural networks suggests that this may also be a promising method for deeper, larger networks.

3. METHOD

In this work, a framework for optimizing the hyper-parameters of a deep network using an evolutionary algorithm is presented. The evolutionary algorithm used is similar to that from [17], but uses a different fitness function. The fitness function used is simply the error on the test set for the dataset used computed after 4000 iterations. Extrapolating the eventual converged performance of the network from early metrics is a current area of research in speeding up hyper-parameter optimization [7]. This work has shown evidence for predicting if a network will be competitive from its early validation results, which is a technique often employed by machine learning experts while hand tuning parameters. The evolutionary algorithm is implemented as a master-slave process where the genes are calculated on a single node (i.e. selection, crossover, mutation are handled by a single node), and many slaves are used to evaluate the fitness of the population.

3.1 Evolutionary Algorithm

Each hyper-parameter to be optimized is encoded as a single gene for each individual. A range and resolution is defined for each gene in order to eliminate searching the areas of the hyper-parameter space that are not of interest. The initial population is generated by sampling each gene from a uniform random distribution, and the fitness for each individual is evaluated. Each generation thereafter is formed using selection, cross-over, and mutation based on the individuals with the highest fitness from the previous generation. This process can be viewed as a single generation of random search that is then followed by results driven search based on the best previous individuals.

Table 1. EA parameters used in this work.

Parameter	Value
N_{GENE}	6
$N_{GENERATIONS}$	35
$N_{INDIVIDUALS}$	500
p_c	0.6
p_m	0.05

Selection is performed by removing any individuals from the population that have a fitness value below the average fitness for their generation. The remaining individuals are then used to create the next generation by performing crossover and mutation. Crossover is performed between consecutive individuals with probability p_c. The crossover point is chosen through a uniform distribution across the range $[0, N_{GENE}]$, where N_{GENE} is the number of genes per individual. Mutation is performed on a gene with probability p_m. Each individual in the resulting child population is then evaluated on a separate node before repeating the process until the maximum number of generations, $N_{GENERATIONS}$, is reached. Table 1 gives the evolutionary algorithm parameters used in this work.

Table 2. Caffe solver parameters used in this work.

Parameter	Value
base_lr	0.001
momentum	0.9
weight_decay	0.004
lr_policy	"fixed"
max_iter	4000

3.2 Computing Framework

ORNL's Titan supercomputer is the computing platform used for this research. The gene for each individual is communicated between nodes using MPI in order for its fitness to be evaluated and its fitness is returned to the master node using MPI. Caffe [11] is used to implement the deep networks. Caffe is a widely used deep learning framework that is highly optimized for speed, comes with many desirable features such as check pointing and is easily extensible.

4. EXPERIMENTS

The CIFAR-10 dataset [14] was used to evaluate the method presented. This dataset consists of 60,000 32x32 pixel color images where the task is to classify the object in the image. There are 10 classes and 10,000 images are designated as a test set. The classification error on this test set is used as the fitness for the corresponding individual.

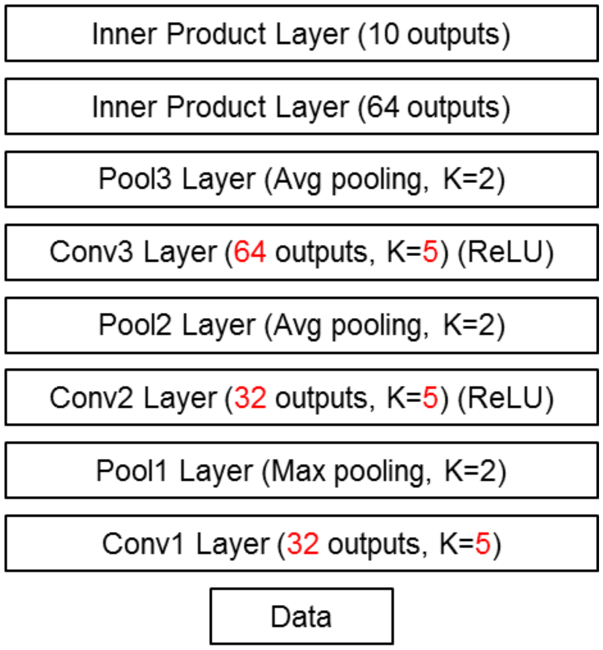
Inner Product Layer (10 outputs)
Inner Product Layer (64 outputs)
Pool3 Layer (Avg pooling, K=2)
Conv3 Layer (64 outputs, K=5) (ReLU)
Pool2 Layer (Avg pooling, K=2)
Conv2 Layer (32 outputs, K=5) (ReLU)
Pool1 Layer (Max pooling, K=2)
Conv1 Layer (32 outputs, K=5)
Data

Figure 1. Network architecture used. Values in red are the defaults for hyper-parameters that are optimized by the evolutionary algorithm.

The network architecture that is used is the default example provided by Caffe and is shown in Figure 1. The hyper-parameters that are optimized are the kernel size and number of filters for each of the convolutional layers. The evolutionary algorithm uses a population size of *500* for each of the *35* generations. The kernel size for the convolutional layers was limited to the range *[1,8]*, and the number of filters was limited to the range *[16,126]*. The hyper-parameters for the solver are given in Table 2.

5. RESULTS

Figure 2 displays the results in a parallel coordinates plot. The results demonstrate that fitness increases between the first generation and the last generation. None of the best performing networks come from the first 10 generations. The kernel size for the convolutional layers displayed much greater importance than the number of filters. Table 3 shows the best hyper-parameters found by the evolutionary algorithm and compares them to the default hyper-parameters. In general the kernel sizes found are smaller than the defaults and the number of filters in each layer is greatly increased. Perhaps unintuitive, the best results came with a much smaller kernel size in the final convolutional layer as compared to the first two convolutional layers (2x2 pixels vs 4x4 pixels). This is in contrast to the default kernel sizes which were the same in all layers. Figure 3 demonstrates the convergence towards these best values as scatter plots of the hyper-parameters versus the generation index.

Table 3. Default Hyper-Parameters vs. Best Hyper-Parameters Found

Parameter	Default	Best 5 Sets
kernel1	5	3-4
output1	32	46-104
kernel2	5	4
output2	32	108-126
kernel3	5	2
output3	64	92-126

6. CONCLUSIONS AND FUTURE WORK

A framework, MENNDL, for optimizing the hyper-parameters of deep networks has been proposed. This tool aims to reduce the difficulty in applying deep networks to new applications and domains by reducing the need to manually iterate over hyper-

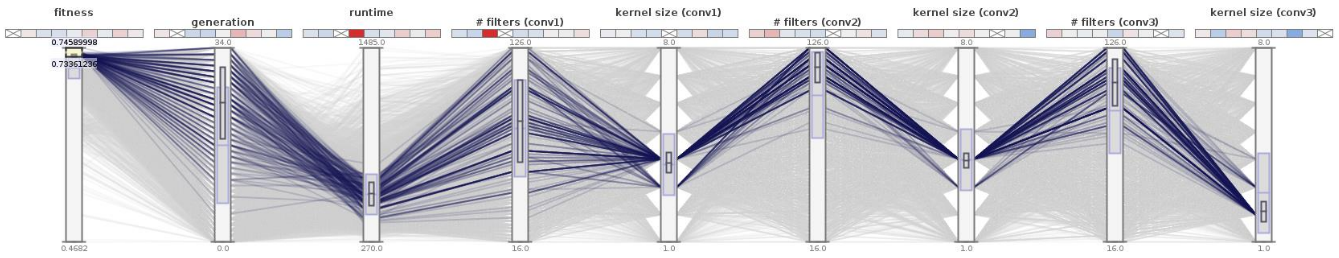

Figure 2. Parallel coordinates plot made using EDEN [19]. Best performing hyper-parameter sets are highlighted in purple.

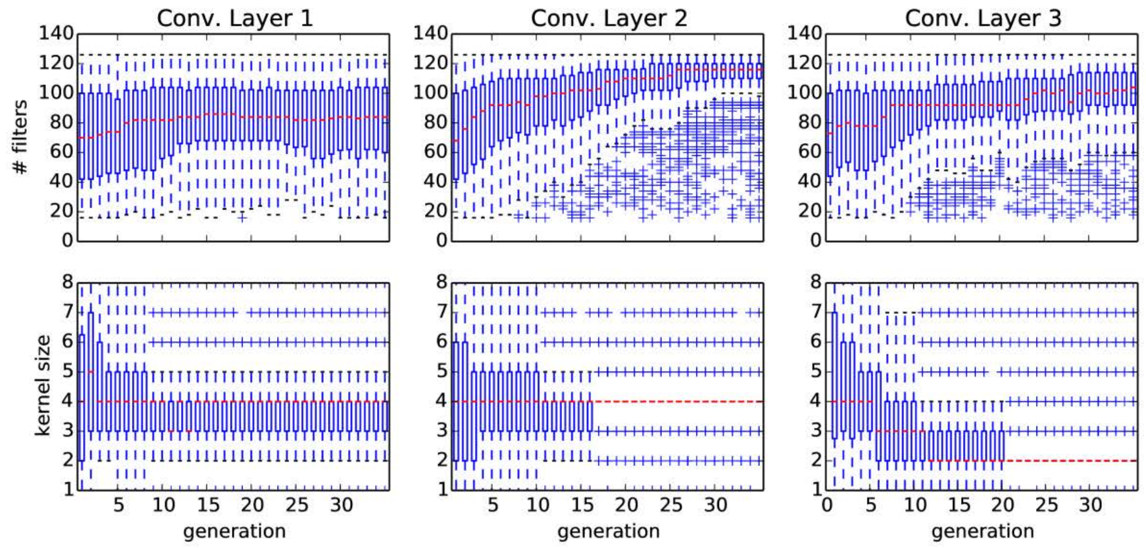

Figure 3. Box plots showing hyper-parameter values versus the generation index.

parameter sets. It can be a more powerful tool than random search since it has the ability to use past results to determine which areas of the hyper-parameter space are searched.

The initial results presented here justify additional exploration in utilizing evolutionary algorithms to aid in the construction of deep networks. An obvious first extension of this work will be to optimize over additional hyper-parameters, including encoding those which cover the layer order, type, and number. Layer modules or groups of commonly adjacent layers (such as convolution - pooling - activation) can be used as an intermediate step toward full design freedom. Although the work presented here most commonly performs mutation by pulling new hyper-parameters from a constrained uniform distribution, it would be interesting to explore layering the genetic algorithm's mutation with other currently researched algorithms in hyper-parameter optimization. For example, a Bayesian optimization method could be learned from performance samples acquired in the course of the genetic algorithm's evolution. Future generations can selectively sample from the distribution learned over the hyper-parameters. Further, utilizing the early termination framework learned in [Domhan] could provide a more sophisticated approach for predicting which network architectures are destined for better performance at convergence. This will be especially important as larger datasets are tackled since they will likely require more complex (and thus higher hyper-parameter count) networks.

The synchronous evolutionary algorithm used here is unable to maximize use of the available resources. Since each set of hyper-parameters will result in differing computational complexities, the time required to evaluate an individual can vary greatly. Thus, many nodes can sit idle while waiting for a single node to finish evaluating its assigned hyper-parameter set. Asynchronous evolutionary algorithms should be explored to maximize use of resources and minimize convergence time.

7. ACKNOWLEDGMENTS

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan(http://energy.gov/downloads/doe-public-access-plan).

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

REFERENCES

[1] Baldi, P. et al. 2014. Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*. 5, (Jul. 2014).

[2] Bergstra, J. and Bengio, Y. 2012. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*. 13, 1 (2012), 281–305.

[3] Bergstra, J. et al. 2013. Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28.

[4] Bergstra, J. et al. 2014. Preliminary evaluation of hyperopt algorithms on HPOLib. International Conference on Machine Learning AutoML Workshop, 2014.

[5] Breuel, T.M. 2015. The effects of hyperparameters on SGD training of neural networks. *arXiv preprint arXiv:1508.02788*. (2015).

[6] Cantú-Paz, E. and Kamath, C. 2005. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *Systems, Man,*

and Cybernetics, Part B: Cybernetics, IEEE Transactions on. 35, 5 (2005), 915–927.

[7] Domhan, T. et al. 2015. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. 24th International Joint Conference on Artificial Intelligence (IJCAI), 2015.

[8] Fiszelew, A. et al. 2007. Finding optimal neural network architecture using genetic algorithms. *Advances in computer science and engineering research in computing science*. 27, (2007), 15–24.

[9] He, K. et al. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *arXiv preprint arXiv:1502.01852*. (2015).

[10] Hinton, G. et al. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*. 29, 6 (Nov. 2012), 82–97.

[11] Jia, Y. et al. 2014. Caffe: Convolutional architecture for fast feature embedding. *Proceedings of the ACM International Conference on Multimedia* (2014), 675–678.

[12] Koutník, J. et al. 2014. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. (2014), 541–548.

[13] Koutník, J. et al. 2014. Online evolution of deep convolutional network for vision-based reinforcement learning. *From Animals to Animats 13*. Springer. 260–269.

[14] Krizhevsky, A. and Hinton, G. 2009. Learning multiple layers of features from tiny images. Citeseer.

[15] Lamos-Sweeney, J. 2012. Deep learning using genetic algorithms. (2012).

[16] Lander, S. 2014. An Evolutionary Method for Training Autoencoders for Deep Learning Networks. University of Missouri–Columbia.

[17] Patton, R.M. et al. 2010. Analysis and Classification of Mammography Reports Using Maximum Variation Sampling. *Genetic and Evolutionary Computation: Medical Applications*. (2010), 112–131.

[18] Snoek, J. et. al, 2012. Practical Bayesian Optimization of Machine Learning Algorithms. *Advances in neural information processing systems 25*, 2012.

[19] Steed, C.A. et al. 2013. Big data visual analytics for exploratory earth system simulation analysis. *Computers & Geosciences*. 61, (Dec. 2013), 71–82.

[20] Tirumala, S.S. Implementation of Evolutionary Algorithms for Deep Architectures.

[21] Verbancsics, P. and Harguess, J. 2015. Image Classification Using Generative Neuro Evolution for Deep Learning. *2015 IEEE Winter Conference on Applications of Computer Vision (WACV)* (Jan. 2015), 488–493.