Neural Networks Designing Neural Networks: Multi-Objective Hyper-Parameter Optimization

Sean C. Smithson Guang Yang Warren J. Gross Brett H. Meyer
Department of Electrical and Computer Engineering
McGill University
Montreal, Canada
{sean.smithson, guang.yang3}@mail.mcgill.ca, {warren.gross, brett.meyer}@mcgill.ca

ABSTRACT

Artificial neural networks have gone through a recent rise in popularity, achieving state-of-the-art results in various fields, including image classification, speech recognition, and automated control. Both the performance and computational complexity of such models are heavily dependant on the design of characteristic hyper-parameters (e.g., number of hidden layers, nodes per layer, or choice of activation functions), which have traditionally been optimized manually. With machine learning penetrating low-power mobile and embedded areas, the need to optimize not only for performance (accuracy), but also for implementation complexity, becomes paramount. In this work, we present a multi-objective design space exploration method that reduces the number of solution networks trained and evaluated through response surface modelling. Given spaces which can easily exceed 10^{20} solutions, manually designing a near-optimal architecture is unlikely as opportunities to reduce network complexity, while maintaining performance, may be overlooked. This problem is exacerbated by the fact that hyper-parameters which perform well on specific datasets may yield sub-par results on others, and must therefore be designed on a per-application basis. In our work, machine learning is leveraged by training an artificial neural network to predict the performance of future candidate networks. The method is evaluated on the MNIST and CIFAR-10 image datasets, optimizing for both recognition accuracy and computational complexity. Experimental results demonstrate that the proposed method can closely approximate the Pareto-optimal front, while only exploring a small fraction of the design space.

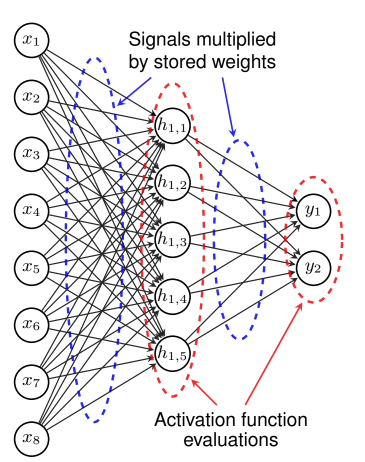
1. INTRODUCTION

Artificial Neural Network (ANN) models have become widely adopted as means to implement many machine learning algorithms and represent the state-of-the-art for many image and speech recognition applications [16]. As the application space for ANNs evolves beyond workstations and data centres towards low-power mobile and embedded platforms, so too must their design methodologies. Mobile voice recognition systems, such as Apple's *Siri*, currently remain too computationally demanding to execute locally on a handset. Instead, such applications are processed remotely and, depending on network conditions, are subject to variations in performance and delay [1]. ANNs are also finding application in other emerging areas, such as autonomous vehicle localization and control, where meeting power and cost requirements is paramount [11].

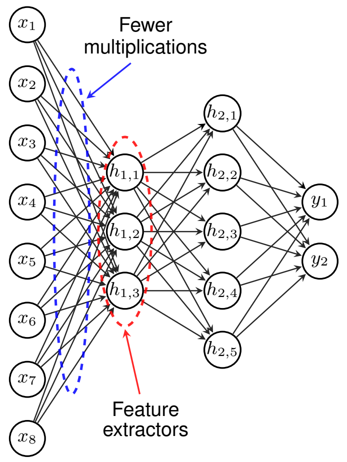
ANNs, which replace manually engineered computer algorithms, must be trained instead of programmed. This training involves an optimization process where network weights are adjusted with the objective of minimizing the output error. These adjustments often involve a variation of the Stochastic Gradient Descent (SGD) method [16]. While these training methods have been automated, much of the design process and choice of network hyper-parameters (e.g., number of hidden layers, nodes per layer, or choice of activation functions) has been historically relegated to manual optimization. This relies on human intuition and expert knowledge of the target application in conjunction with extensive trial and error [7, 22]. This process is difficult, considering the vast network hyper-parameter space which includes: the number of convolutional or hidden layers, the quantity of nodes in each layer, the type of nonlinear activation functions, and many others which depend on the system in-hand. In addition, the problem with manual hyper-parameter tuning is that there is no guarantee that the process will result in optimal configurations. Not only does the diversity of possible hyper-parameters create extremely large design spaces, but time intensive training phases on comprehensive data sets must also be performed prior to evaluating candidate solutions. This significant computational overhead renders exhaustive searches intractable, and necessitates the use of automated Design Space Exploration (DSE) tools to intelligently explore the solution space while limiting the number of candidate models that must be trained and evaluated.

With the proliferation of machine learning on embedded and mobile devices, ANN application designers must now deal with stringent power and cost requirements [9, 16, 21]. These added constraints transform hyper-parameter design into a multi-objective optimization problem where no single optimal solution exists. Instead, the set of points which are not dominated by any other solution forms a Pareto-optimal front. Simply put, this set includes all solutions for which

ICCAD '16, November 07 - 10, 2016, Austin, TX, USA
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-4466-1/16/11…$15.00
DOI: http://dx.doi.org/10.1145/2966986.2967058


Signals multiplied by stored weights
Activation function evaluations

(a) Initial configuration with a single hidden layer



(b) Reduction in the first stage of multiplications

Figure 1: Example MLP configurations

no other is objectively superior in all criteria. Formally, a solution vector (a_1, b_1) (where a and b are two objectives for optimization) is said to dominate another point (a_2, b_2) if $a_1 < a_2$ and $b_1 \leq b_2$, or $b_1 < b_2$ and $a_1 \leq a_2$; the set of points which are not dominated by any other solution constitutes the Pareto-optimal front [19].

This paper presents an automated DSE method that effectively trains a neural network to design other neural networks, optimizing for both performance and implementation cost. This meta-heuristic ANN exploits machine learning to predict the performance of candidate solutions (modelling the response surface); the algorithm learns which points to explore and avoids the lengthy computations involved in evaluating solutions which are predicted to be unfit. This leveraging of Response Surface Modelling (RSM) techniques dramatically reduces the proposed algorithm run-time, which can ultimately result in the reduction of product design time, application time-to-market, and overall Nonrecurring Engineering (NRE) costs.

1.1 Motivational Example

While there are many different ANN models, the Multi-Layer Perceptron (MLP) is a well-known form, which rose in popularity with the advent of the back-propagation training algorithm [5]. Characterized by a series of cascaded *hidden* layers, MLPs have a single feed-forward path from the input to output layers. MLP layers are also fully-connected; each node has a connection to each and every node in adjacent layers. When illustrated as a directed graph (shown in Figure 1), graph connections are representative of signals being multiplied by corresponding weights, and graph nodes the summation of all inputs followed by non-linear activation functions. The evaluation of such elements, commonly Rectified Linear Units (ReLUs) or sigmoid functions, is comparatively simple. Therefore, multiply-accumulate operations and memory accesses remain the dominant tasks in terms of cost [9, 12].

While structurally simple, determining an optimal MLP configuration is a difficult task due to the large number of parameters inherent to even the simplest designs. For example, given the simple MLP shown in Figure 1(a), with i inputs, a single hidden layer with j nodes, and k nodes in the output layer, $i \times j + j \times k$ operations must be evaluated at each inference pass and an equal number of weights must be accessed in memory. Starting from this initial configuration, a designer may then choose to include a second hidden layer, as illustrated in Figure 1(b); the first will then act as a feature extractor and the newly added layer will then process only the limited number of features generated by the first. This alteration allows for the reduction in dimension of the first hidden layer, which reduces the total number of connections to the input layer. However, this also increases the network depth and results in a cost penalty (in terms of requiring additional memory accesses and arithmetic operations associated with the newly added layer). The key problem demonstrated is that even for these two simple configurations, there is no systematic way to determine which design yields superior performance without having trained and evaluated both. Even when the designer has *a priori* knowledge of the application, determining the optimal hyper-parameters is non-intuitive, especially for deep networks.

A concrete example of the described problem would be the design of an embedded system to recognize handwritten numerical characters (such as those contained in the MNIST dataset). If the implementation goal is throughput of categorized digits, and a penalty is incurred when a character is misclassified (perhaps requiring manual intervention), then a smaller network that requires fewer clock cycles to evaluate may still result in an overall throughput exceeding those of more accurate alternatives. In such a scenario, the engineer would require knowledge of the cost and performance of all Pareto-optimal solutions in order to meet all requirements with the lowest implementation costs.

Even more complex structures are those of Convolutional Neural Network (CNN), which have demonstrated state-of-the-art results in image recognition [15]. The use of convolutional layers further complicates the design process as they introduce a separation between the memory and processing requirements. Illustrated in Figure 2, convolutional layers are composed of trainable filters (five *3-by-3* kernels

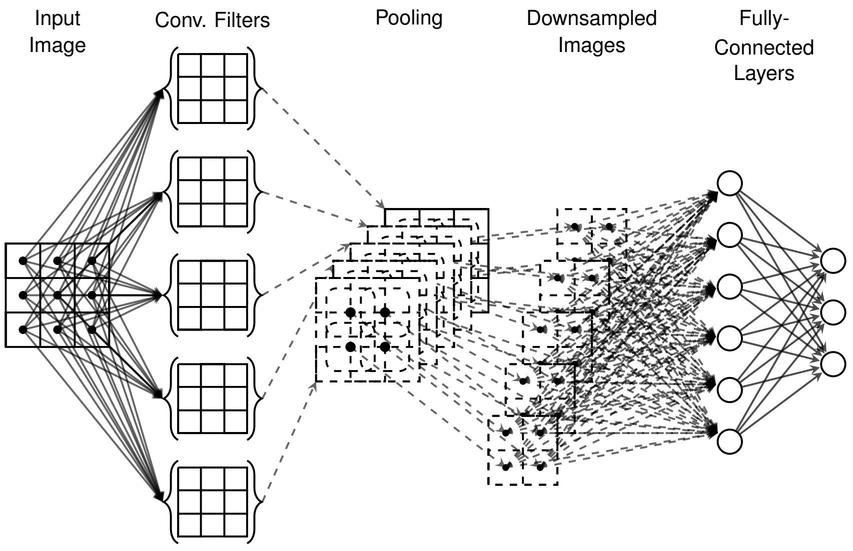


Figure 2: CNN computational stages

are shown). CNNs have the advantage of reduced memory requirements due to each convolutional filter reusing the same kernel weights for all input values. However, this is at the expense of increased processing demands, the result of each convolutional filter requiring N^2 multiplication operations for each input value (where the convolutional kernels are sized N-*by*-N). In a CNN, each filter produces a processed copy of the input data (as illustrated in Figure 2); and without any form of down-sampling, this greatly increases the computational complexity of the following layers. An example of a possible efficient down-sampling method is the inclusion of max-pooling (or mean-pooling) after convolutional layers [15]. Illustrated in Figure 2, max-pooling refers to partitioning the filtered images into non-overlapping K-*by*-K regions, with each outputting the maximum pixel value within (alternatively, mean-pooling would output the mean value). All of these additional CNN parameters further increase the design space dimensionality, forcing a designer to not only choose how many filters to use in each layer, but also the kernel and pooling sizes. This greatly affects both the performance and computational complexity of the resulting networks.

1.2 Summary of Contributions

The key contributions of this work are presented as follows:

- We introduce a DSE method, which employs RSM techniques to predict classification accuracy, to automate the design of ANN hyper-parameters. This method is then validated with MLP and CNN designs targeting the CIFAR-10 and MNIST image recognition datasets [14, 17].
- We demonstrate that multi-objective hyper-parameter optimization can successfully be used as a method to reduce ANN implementation cost (computational complexity).

2. RELATED WORK

This work exists at the intersection of two fields: automated hyper-parameter optimization, and reduction of ANN computational complexity. To our knowledge, this work is the first that has applied automated hyper-parameter optimization as a method of reducing ANN computational complexity.

2.1 Hyper-Parameter Optimization

The design of neural network hyper-parameters has long been considered to be unwieldy, unintuitive, and as a consequence, ideal for automated hyper-parameter optimization techniques such as in [3], [4], [7], and [22]. However, these works have been developed with the sole purpose of optimizing performance, with little regard to the resulting computational resource requirements.

Two types of sequential hyper-parameter optimization algorithms were presented in [4]; in both cases experimental results compared positively to human designed alternatives. These findings were echoed in [3], where it was demonstrated that a random search of a large design space, which contains areas that may be considered less promising, can at times exceed the performance of manually tuned hyper-parameters.

Positive results were also presented in [7] where similar algorithms were extended using a method to extrapolate the shape of learning curves during training, so as to evaluate fewer epochs for unfit solutions and reduce design time. Since parameters which perform favourably for a large network may not be optimal for a smaller alternative, and the most important hyper-parameters (with the greatest impact on resulting performance) may vary for different data sets, the need for multi-objective optimization (especially where low power platforms are targeted) is clear [3, 22].

Additionally, in [20] an automated DSE method was applied to the multi-objective optimization problem of application-specific MPSoC design, a field which also consists of high-dimensionality solution spaces. It was demonstrated that through RSM the presented DSE method could efficiently identify Pareto-optimal architectures.

2.2 Weight Quantization and Pruning

There also exists a body of work attempting to reduce the computational complexity of ANN models through weight quantization or the removal of extraneous node connections (pruning). The research in [6] and [9] are examples of two methods that construct networks which reduce the need for multiplications, or modify already trained networks in order to minimize the number of non-zero weights.

In [6], the authors attempted to reduce the computational resources required when evaluating fully-connected, as well as convolutional, networks through representing all weights as binary values of $+1$ or -1. Doing so reduces the number of multiplication operations performed each forward pass; however the requirement to store floating-point weights during training remains. The work in [6] also compared trained binary weighted networks to traditional equivalents with equal layer dimensions and similar performance was demonstrated. However, [6] only considered very large network dimensions; further benefits may be obtained from smaller optimized architectures.

Instead of restricting weights to specific values, in [9] a pruning method was presented in which a network can be trained while reducing the number of non-zero weights. The resulting compressed networks have lower bandwidth requirements and require fewer multiplications due to most weights being zero. The results in [9] demonstrated up to 70% reductions in the numbers of floating-point operations required for various networks, with little to no reduction in performance. However, such pruning methods are not mutually exclusive to the use of DSE tools and could very well be implemented in conjunction with the presented methodology in order to compress an already optimized network configuration.

3. ANN SELF-DESIGN METHODOLOGY

This work presents a DSE approach that searches for Pareto-optimal hyper-parameter configurations and has been applied to both MLP and CNN topologies. The design space is confined to: the numbers of Fully-Connected (FC) and convolutional layers, the number of nodes or filters in each layer, the convolution kernel sizes, the max-pooling sizes, the type of activation function, and network training rate. These degrees of freedom constitute vast design spaces and all strongly influence the resulting networks' cost and performance.

For design spaces of such size, performing an exhaustive search is intractable (designs with over 10^{10} to 10^{20} possible solutions are not uncommon), therefore we propose to model the response surface using an ANN for regression where the set of explored solution points is used as a training set. The presented meta-heuristic ANN is then used to predict the performance of candidate networks; only points which are expected not to be Pareto-dominated are explored.

3.1 Main DSE Algorithm Overview

A flowchart describing the DSE implementation is shown in Figure 3. The general steps performed during the design space exploration can be broken down into:

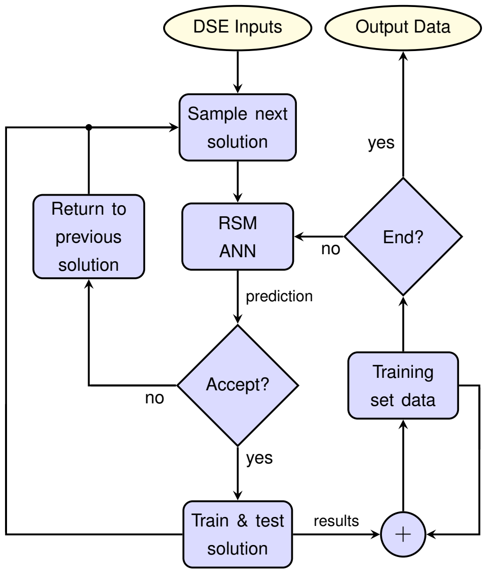


Figure 3: DSE algorithm flow

1. Sample the next candidate point from a Gaussian distribution centred around the previously explored solution (or sample a random point for the first iteration).
2. Predict the candidate solution performance using the RSM neural network, and calculate the cost as:

$$cost = (\#\text{ of weights}) \times (\text{weight unit cost}) + \\ (\#\text{ of multiplications}) \times (\text{multiplication unit cost})$$

3. Compare the predicted results to the current Pareto-optimal front. If the candidate is predicted to be Pareto-dominated, it is accepted with probability α, otherwise it is accepted with probability $1 - \alpha$.
4. If rejected, the previously explored solution is rolled back and the algorithm returns to Step 1.
5. If accepted, the candidate model is trained, tested, and the evaluated results are added to the training set of the RSM ANN (which is then retrained).
6. Finally, if the training set size exceeds the maximum number of desired iterations, the process ends. Otherwise, the algorithm returns to Step 1 and a new solution is sampled.

3.2 Candidate Solution Sampling

The sampling strategy proposed is an adaptation of the Metropolis-Hastings algorithm [18]. In each iteration a new candidate is sampled from a Gaussian distribution centred around the previously explored solution point. Performing this random walk limits the number of samples chosen from areas of the design space that are known to contain unfit solutions, thereby reducing wasted exploration effort. However, exploring an inferior solution may eventually lead to those of superior performance, therefore the probability of accepting such a solution (α) must remain greater than zero; this also ensures that the training set for the RSM ANN remains varied. All experimental results in Section 5 were obtained with $\alpha = 10^{-4}$.

3.3 Predictive Neural Network Design

We choose to model the response surface using a MLP model with an input set representative of network hyper-parameters and a single output trained to predict the error of corresponding networks. This RSM ANN is composed of two hidden ReLU layers and a linear output layer. Experimental results demonstrated that sizing the hidden layers with 25× to 30× the number of input nodes provided the best performance.

The RSM network inputs are formed as arrays characterizing all explored dimensions. Integer input parameters (such as number of nodes in a hidden layer, or size of the convolutional kernels) are scaled by the maximum possible value of the respective parameter, resulting in normalized variables between 0 and 1. For each parameter that represents a choice where the options have no numerical relation to each other (such as whether ReLU or sigmoid functions are used) an input mode is added and the node that represents the chosen option is given an input value of 1, all others -1. For example, a solution with two hidden layers with 20 nodes each (assuming a maximum of 100), using ReLUs (with the other option being sigmoid functions) and with a learning rate of 0.5 would be presented as input values: $[0.2, 0.2, 1, -1, 0.5]$.

The RSM model was trained using SGD, where 100 training epochs were performed on the set of explored solutions each time the next is evaluated (and in turn, added to the training set). The learning rate was kept constant, with a value of 0.1, in order to train the network quickly during early exploration, when the set of evaluated solutions is limited.

4. EXPERIMENTAL SETUP

We evaluated our DSE strategy on ANN applications targeting the standard MNIST and CIFAR-10 image recognition datasets. In the case of designing MLP models targeting the simpler MNIST problem, the results generated by the DSE algorithm were compared to the true Pareto-optimal front obtained from an exhaustive search performed on a constrained solution space. Such a limitation of the design space was required in order to render the exhaustive search tractable. The DSE algorithm was also evaluated on much larger design spaces for both MLP and CNN models targeting MNIST as well as an even larger one for the design of CNNs targeting CIFAR-10. In all cases only a few iterations were required in order to converge to approximated Pareto-optimal fronts.

In order to perform the DSE, all ANN models were trained and tested using the *Theano* framework in order to take advantage of GPGPU optimizations [2]. This allowed the entire DSE and evaluation to be performed using a Nvidia *Tesla K20c* GPU. Upon completion, all explored network data and simulation results were stored to disk, allowing future design re-use.

To model cost, we assumed normalized memory access and multiply-accumulate operation costs. These can be quantified at either the software or hardware levels (depending on the target application). In both cases the costs (such as power, area, or time) are implementation specific and the exact values used are transparent to the DSE algorithm.

Table 1: Normalized cost values

Operation	Energy Cost from [9]	Normalized Cost
Addition	0.9pJ	N/A
Multiplication	3.7pJ	N/A
Multiply-Accumulate	0.9pJ + 3.7pJ	1
DRAM Memory Access	640pJ	139

Normalized costs assumed for all the experimental results are shown in Table 1, which are based on the energy costs for 32-bit floating-point operations from [9].

We experimentally validated the DSE algorithm on two separate image recognition databases; future work is planned to address a broader range of benchmarks. The first, MNIST, contains 28×28 pixel grayscale images of handwritten numeric characters (from 0 to 9). The second, CIFAR-10, is a much more complicated dataset composed of 32×32 RGB colour images, each belonging to one of ten object categories.

5. RESULTS

The network hyper-parameters composing the design spaces explored are outlined in Table 2. In addition, several parameters were kept constant for all experiments: output layers were composed of *softmax* units (with *jth* output defined as $e^{in_j} \div \sum_{k=1}^{K} e^{in_k}$ for a layer with K nodes) and all network training was performed with categorical cross-entropy ($-\sum [targets \times log\,(predictions)]$) as loss function [8, 12]. Finally, for all except the reduced MNIST design problem, batch normalization was included after each network layer in order to smooth the response surfaces [10].

5.1 Exhaustive Search Comparison

In order to evaluate the efficiency with which the method approximates the true Pareto-optimal front, we first compare experimental results to those of an exhaustive search targeting the design of MLP models for the MNIST dataset. In order to make an exhaustive search tractable, we limited the design space to only the values outlined in the first section of Table 2. This resulted in a moderate design space of 10^4 solutions, all of which were trained and tested.

The results of executing the DSE algorithm for 200 iterations (each iteration represents a design training, evaluation, and model update pass) are plotted alongside those of the exhaustive search in Figure 4. These results demonstrate that the true Pareto-optimal front is very closely approximated by the presented method, while requiring very few solutions to be evaluated. However, it should be noted that SGD is by nature non-deterministic and training the same network twice may yield different performances. Consequently, the DSE generated results (shown in Figure 4) dominate those of the exhaustive search at several points. This figure also demonstrates that the majority of solutions evaluated by the DSE algorithm remain within a close vicinity of the true Pareto-front, successfully avoiding Pareto-dominated areas. Since the true Pareto-optimal front is

Table 2: Experimental design-space parameters

Target	Parameter	Range	Steps	Scale
Restricted MNIST (MLP)	Number of FC layers	1 to 3	3	Linear
	Nodes per FC layer	10 to 200	10	Log
	Learning rate	0.01 to 0.8	9	Log
	Activation function	ReLU	1	N/A
	Training algorithm	SGD	1	N/A
	Batch sizes	200	1	N/A
	Training epochs	10	1	N/A
MNIST (MLP)	Number of FC layers	1 to 3	3	Linear
	Nodes per FC layer	10 to 500	85	Log
	Learning rate	0.001 to 0.8	16	Log
	Activation function	ReLU or tanh	2	N/A
	Training algorithm	SGD	1	N/A
	Batch sizes	200	1	N/A
	Training epochs	10	1	N/A
MNIST (CNN)	Number of CNN layers	1 to 2	2	Linear
	Number of FC layers	1 to 2	2	Linear
	Filters per CNN layer	8 to 128	8	Log
	Nodes per FC layer	10 to 250	16	Log
	Learning rate	0.01 to 0.8	12	Log
	Filter kernel size	1x1 to 5x5	3	Linear
	Max-pool size	2x2 to 4x4	3	Linear
	Activation function	ReLU	1	N/A
	Training algorithm	SGD	1	N/A
	Batch sizes	200	1	N/A
	Training epochs	10	1	N/A
CIFAR-10 (CNN)	Number of CNN layers	1 to 3	3	Linear
	Number of FC layers	1 to 2	2	Linear
	Filters per CNN layer	16 to 512	20	Log
	Nodes per FC layer	10 to 500	16	Log
	Learning rate	0.001 to 0.1	16	Log
	Filter kernel size	3x3 to 9x9	4	Linear
	Max-pool size	2x2 to 3x3	2	Linear
	Activation function	ReLU	1	N/A
	Training algorithm	*Adam* [13]	1	N/A
	Batch sizes	200	1	N/A
	Training epochs	20	1	N/A

known for this restricted case, we use Average Distance to Reference Set (ADRS) as the metric of evaluation; quantifying how closely the approximated set differs from the exact [20]. ADRS is defined in Eq. (1), where Λ and Π are the approximate and exact Pareto-optimal fronts, and the $\delta\left(x_R, x_A\right)$ function represents the normalized distance between solutions x_R and x_A.

$$ADRS(\Pi, \Lambda) = \frac{1}{|\Pi|} \sum_{x_R \in \Pi} \min_{x_A \in \Lambda} \Big(\delta\left(x_R, x_A\right) \Big) \qquad (1)$$

The evolution of the approximated Pareto-optimal set discovered by the DSE algorithm, as a function of iterations completed is plotted in Figure 5(a). Evident from this figure is that the optimal front obtained from the DSE algorithm progressively approaches the true Pareto-optimal, while evaluating only a comparatively small number of iterations. The proposed method identifies the optimal solutions with high accuracy, achieving low ADRS values of 7.1% after completing only 50 iterations, 5.0% after 100, and 3.6% after 200. The results in Table 3 also demonstrate fast convergence, with the largest changes occurring over the first 30 iterations. Further execution yields more gradual changes as the approximated Pareto-front asymptotically approaches the exact. Decreasing design time (both in terms of computation and man-hours), these results also demonstrate that by exploring less than 1% of the design space, the DSE method closely predict which hyper-parameters are required for optimal solutions. In addition, the generated set of Pareto-optimal configurations also exposes the trade-offs application designers may want to make for cost-constrained systems, allowing for more informed design choices to be made.

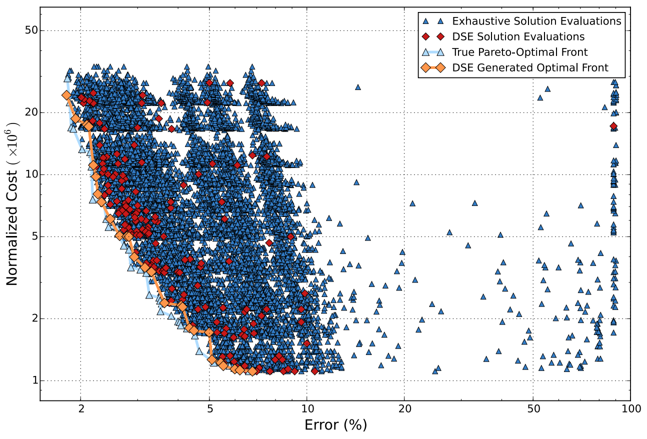


Figure 4: Exhaustive search versus DSE results

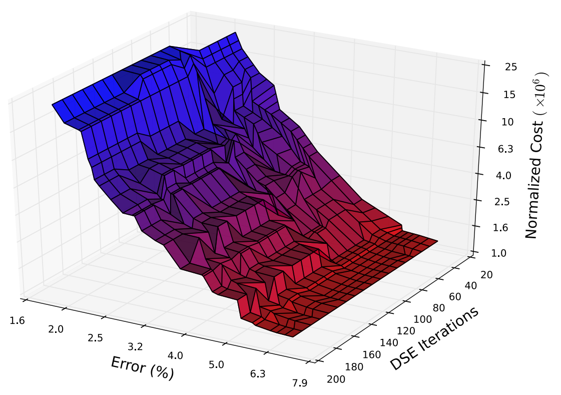
Table 3: Evolution of ADRS values

DSE Iterations	ADRS of Explored Set	Execution Time
10	30%	2.0 min
20	21%	4.1 min
30	8.4%	6.4 min
50	7.1%	11.2 min
70	6.1%	16.4 min
100	5.0%	25.4 min
150	4.5%	45.2 min
200	3.6%	70.1 min

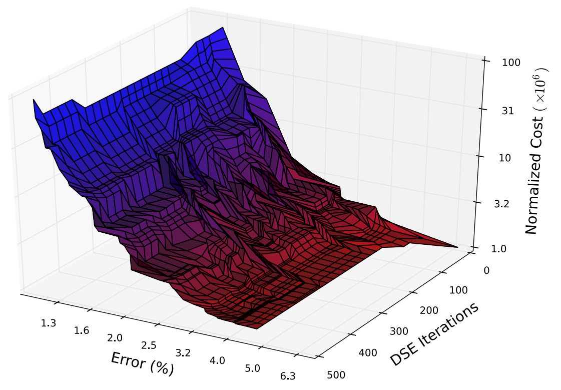
5.2 Evaluation on Expanded Design Spaces

In order to evaluate performance of the heuristic method for much larger designs, the algorithm was run on the remaining spaces described in Table 2 for both MLP and CNN design problems. The total execution times (on an Intel *Xeon E5-1603* CPU with 8GB of RAM and a Nvidia *Tesla K20c* GPU) for the design examples are listed in Table 4 and the corresponding Pareto-optimal results (plotted as functions of the number of iterations completed) are shown in Figure 5. In these plots, the colour scheme presents solutions with low error in blue, and low cost in red. Even though the DSE outputs cannot be directly compared to the true results from an exhaustive search (the CIFAR-10 example design space exceeds 10^{10} solutions), the trends discussed in Section 5.1 are mirrored by all plots in Figure 5.

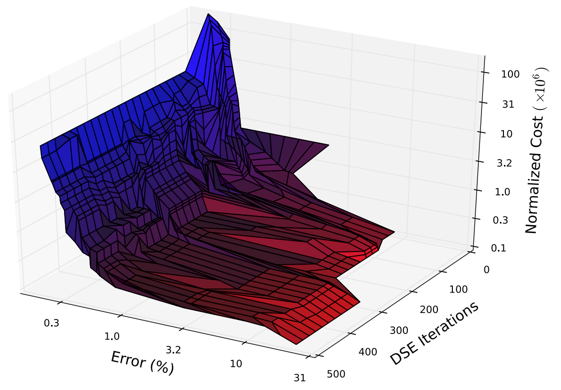
Because of the intractable nature of such exhaustive searches, the DSE generated results are not expected to always predict the true Pareto-optimal fronts. Instead, automated explo-



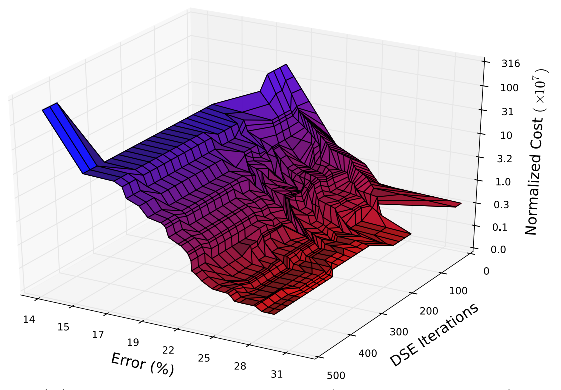
(a) Pareto-optimal results (restricted MNIST MLP design-space)



(b) Pareto-optimal results (MNIST MLP)



(c) Pareto-optimal results (MNIST CNN)



(d) Pareto-optimal results (CIFAR-10 CNN)

Figure 5: DSE generated experimental results

Table 4: DSE execution times

Target	DSE Algorithm Execution Time	Mean Solution Evaluation Time
MNIST (MLP)	3.6 h	30 s
MNIST (CNN)	18 h	2 min
CIFAR-10 (CNN)	66 h	8 min

ration must only match, or exceed, the fitness of manually designed networks in order to provide NRE cost reductions. In comparison with manually designed architectures in literature, the Pareto-optimal results in Figure 5(c) include points that offer equivalent performance to the CNN designs in [12] and [17], with implementation costs as low as 25% of their manually designed counterparts (when weighted with the same cost model detailed in Table 1). This demonstrates that the proposed DSE method can design ANNs with vastly reduced cost requirements and same levels of performance when compared to manual design approaches.

Furthermore, the Pareto-optimal results can also find data points with substantial cost savings penalized only by a small decrease in performance. Given that increasing accuracy by only 0.01% requires a doubling in implementation cost (for MNIST CNN designs shown in Figure 5(c)), these additional points are invaluable for application platforms with extremely stringent cost budgets.

5.3 RSM Prediction Accuracy

In order to validate the assumption that a neural network can be trained through regression to model the response surface with sufficient accuracy, the RSM ANN prediction error is plotted in Figure 6. This graph plots the absolute value of the error (% difference between the predicted and the evaluated performance of each explored solution) for each of the 500 DSE algorithm iterations performed during the MNIST CNN design example (with results in Figure 5(c)). The narrow error spikes, which are expected, occur at points where the DSE algorithm encounters previously unexplored areas. As these solutions are added to the training set, the prediction error decreases as the response surface model is updated, and the spikes begin to occur less frequently. Outside of these sparse peaks, the prediction accuracy is exceptionally high; the mean error over the last 95 iterations (all points after the last spike) is only 0.35%.

6. CONCLUSION

A DSE method to automate the multi-objective optimization of neural network hyper-parameters, reducing both algorithm error and computational complexity, was presented. When compared to the results of an exhaustive search on a restricted design space targeting the MNIST dataset, the presented method was shown to dramatically reduce computation time required for convergence to a Pareto-optimal solution set. A low ADRS of 5% was achieved after only 100 iterations; in practice, fewer solution evaluations may be required, with corresponding execution times less than those in Table 4. Furthermore, scalability of the method was demonstrated on larger design spaces for both MLP and CNN models targeting the CIFAR-10 dataset as well.

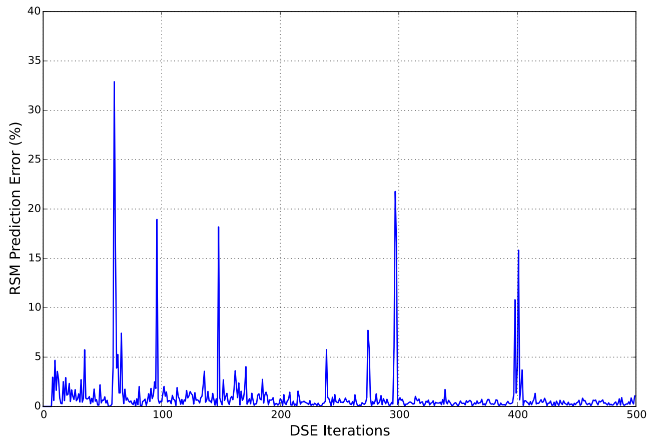


Figure 6: RSM error during MNIST CNN design

Even when evaluated on massive design spaces, the presented DSE method was found to still efficiently converge to a diverse Pareto-optimal front. Not only was the automation of ANN hyper-parameter design process demonstrated to be both feasible and time efficient, but when compared to manually designed networks from literature, the automated DSE technique produced results with near identical performance while reducing the associated costs by a factor of 3×. As applications for ANNs make further inroads in mobile and embedded market segments, the need to reduce time-to-market and NRE costs will necessitate the use of such automated design methods.

7. ACKNOWLEDGEMENTS

This work was supported in part by a Postgraduate Scholarship - Doctoral (PGS-D) scholarship from the Natural Sciences and Engineering Research Council of Canada (NSERC), as well as equipment donations from Nvidia Corporation.

8. REFERENCES

[1] M. Assefi, M. Wittie, and A. Knight. Impact of network performance on cloud speech recognition. In *Computer Communication and Networks (ICCCN), 2015 24th Int. Conf.*, pages 1–6, Aug 2015.

[2] F. Bastien et al. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

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

[4] J. S. Bergstra et al. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems 24*, pages 2546–2554. Curran Associates, Inc., 2011.

[5] R. Collobert and S. Bengio. Links between perceptrons, MLPs and SVMs. In *Proc. of the Twenty-first Int. Conf. on Machine Learning*, ICML '04, pages 23–30, New York, NY, USA, 2004. ACM.

[6] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc., 2015.

[7] T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proc. of the Twenty-Fourth Int. Joint Conf. on Artificial Intelligence, IJCAI*, pages 3460–3468, July 2015.

[8] A. Graves, A. R. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pages 6645–6649, May 2013.

[9] S. Han et al. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems 28*, pages 1135–1143. Curran Associates, Inc., 2015.

[10] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[11] S. Ishibushi et al. Statistical localization exploiting convolutional neural network for an autonomous vehicle. In *Industrial Electronics Society, IECON 2015 - 41st Annual Conf. of the IEEE*, pages 1369–1375, Nov 2015.

[12] K. Jarrett et al. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th Int. Conf.*, pages 2146–2153, Sept 2009.

[13] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[14] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[17] Y. LeCun et al. Gradient-based learning applied to document recognition. *Proc. of the IEEE*, 86(11):2278–2324, Nov 1998.

[18] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, Jun 1953.

[19] T. Okabe, Y. Jin, and B. Sendhoff. A critical survey of performance indices for multi-objective optimisation. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 2, pages 878–885, Dec 2003.

[20] G. Palermo, C. Silvano, and V. Zaccaria. ReSPIR: A response surface-based Pareto iterative refinement for application-specific design space exploration. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Trans.*, 28(12):1816–1829, Dec 2009.

[21] E. Park et al. Big/little deep neural network for ultra low power inference. In *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2015 Int. Conf.*, pages 124–132, Oct 2015.

[22] S. R. Young et al. Optimizing deep learning hyper-parameters through an evolutionary algorithm. In *Proc. of the Workshop on Machine Learning in High-Performance Computing Environments*, MLHPC '15, pages 4:1–4:5, New York, NY, USA, 2015. ACM.