Computing Nonlinear Active Subspaces for Highly Parameterized Optimization Problems using Autoencoders

Gabriele Boncoraglio, SUNet ID: gbonco

Abstract

A novel approach to solving highly parameterized optimization problems is introduced specifically related to problems involving running expensive computer simulations such as aircraft design. These optimization problems require running expensive computational fluid dynamics (CFD) simulations in order to evaluate objective function and constraints. In particular, very often, the number of computer simulation increases when the optimization parameter space is high-dimensional making the problem computationally intensive. Various approaches have been proposed in literature to reduce the dimensionality of the problem, such as the method of active subspace which consist in a linear approximation of the subspace. This method is not suitable when the subspace is nonlinear. The approach proposed consists of using an autoencoder in order to have a nonlinear approximation of the subspace. Different autoencoder architectures are compared in performances and accuracy. Finally, the proposed approach is applied first to a series of unconstrained optimization problems and finally to the design optimization of the mAEWing2 aircraft. The preliminary results show that an autoencoder can efficiently represent a nonlinear manifold and reduce the dimensionality of the problem allowing to solve the optimization problem is a subspace and reduce the number of iteration for convergence.

1 Introduction

MultiDisciplinary Optimization (MDO) problems arise in many engineering applications such as design of an aircraft and involves running very computationally expensive simulations. These optimization problems are challenging when **the number of optimization parameters is relatively large** and the model involves an expensive computer simulation. Therefore, one possible approach is to reduce the number of parameters for the optimization problem, solving the optimization problem on a low-dimensional subspace and permitting otherwise infeasible studies. One technique used to compute this subspace is the method of Active Subspace (AS)[1]: many multivariate functions in engineering models vary primarily along a few directions in the space of input parameters. The method of active subspaces detects the directions of the strongest variability using evaluations of the gradient and subsequently exploits these directions to solve an optimization problem on a low-dimensional subspace. The key components of active subspace methods are the left singular vectors of $\mathbf{M}$, a matrix whose elements are partial derivatives of the simulation's input/output map. Let us consider a scalar objective function $f(\mathbf{x}) \in \mathbb{R}$ of a vector $\mathbf{x} \in \mathbb{R}^{N_x}$, and its gradient $\nabla f(\mathbf{x}) \in \mathbb{R}^{N_x}$. First, the gradient of f at m points needs to be sampled: $\{\nabla f(\mathbf{x}_1), \cdots, \nabla f(\mathbf{x}_i), \cdots, \nabla f(\mathbf{x}_m)\}$. Then the matrix $\mathbf{M}$ contains a set of m pre-computed gradients in the parameter space and its SVD can be computed:

$$\mathbf{M} = \left[\nabla f(\mathbf{x}_1), \cdots, \nabla f(\mathbf{x}_m)\right] = \left[\mathbf{U}_x \;\; \mathbf{U}_{tr}\right] \begin{bmatrix} \mathbf{\Sigma}_x & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}_{tr} \end{bmatrix} \begin{bmatrix} \mathbf{V}_x \\ \mathbf{V}_{tr} \end{bmatrix} \tag{1}$$

with $\mathbf{U}_x \in \mathbb{R}^{N_x \times n_x}$, where n_x is the dimension of the low-dimensional active subspace. Selecting the right n_x dimension of the active subspace is not a easy task. Usually, looking the singular values, the dimension of $\mathbf{U}_x$ is determined by choosing the n_x biggest singular values. Finally a low-rank active subspace of the input parameters is found:

$$\mathbf{x} \approx \mathbf{U}_x \mathbf{x}_r \tag{2}$$

where $\mathbf{x}_r$ is the reduced set of coordinates in the active subspace, with $\mathbf{x}_r \in \mathbb{R}^{n_x}$. The challenge is that AS does not always work for every problem. In fact, since the approximation in (2) is a linear approximation, it cannot capture nonlinear effects. Hence, the idea of this paper is to use a non-linear approximation for the active subspace. In order to do this, the idea is to use an autoencoder. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction. Hence, the idea, is to learn a low-dimensional representation of the gradient using a nonlinear function g which comes from the decoder of the autoencoder and using the following non-linear approximations for the input parameters:

$$\mathbf{x} \approx g(\mathbf{x}_r) \tag{3}$$

The method of active subspace uses the gradients $\{\nabla f(\mathbf{x}_1), \cdots, \nabla f(\mathbf{x}_m)\}$ to learn a **linear low-dimensional representation** of the input parameters $\mathbf{x} \approx \mathbf{U}_x \mathbf{x}_r$. On the other hand, the idea of this paper is to use an autoencoder and the gradients $\{\nabla f(\mathbf{x}_1), \cdots, \nabla f(\mathbf{x}_m)\}$ to learn a **nonlinear low-dimensional representation** of the input parameters $\mathbf{x} \approx g(\mathbf{x}_r)$. Once the low-dimensional active subspace is computed, the optimization problem is solved in the subspace.

2 Design optimization problem

The aim of this paper is solving *efficiently* highly parameterized optimization problems such as the design optimization of the mAEWing2 aircraft. This optimization problem has the following definition:

$$
\begin{aligned}
\underset{\mathbf{x}\in\mathbb{R}^{33}}{\text{maximize}} \quad & \left(\frac{L(\mathbf{x})}{D(\mathbf{x})}\right)^2 \\
\text{subject to} \quad & \frac{L(\mathbf{x}) - \frac{3}{4}L_0}{L_0} \geq 0 \\
& \mathbf{x}^{ub} \leq \mathbf{x} \leq \mathbf{x}^{ub}
\end{aligned} \tag{4}
$$

where $L(\mathbf{x})$ and $D(\mathbf{x})$ are respectively the lift and drug generated by the aircraft. L_0 is the lift generated by the initial configuration of the wing and $\mathbf{x}^{ub}$ and $\mathbf{x}^{lb}$ are the upper and lower bounds for the optimization parameters. The set of 33 parameters $\mathbf{x} \in \mathbb{R}^{33}$ modify the shape of the wing. Figure (1) and (2) describe in more details these 33 parameters.

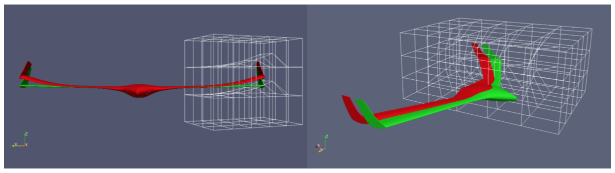

Figure 1: Left: $\{x^1, x^2\}$ modify the dihedral angle of the wing. Right: $\{x^3\}$ modifies the sweep angle of the wing.

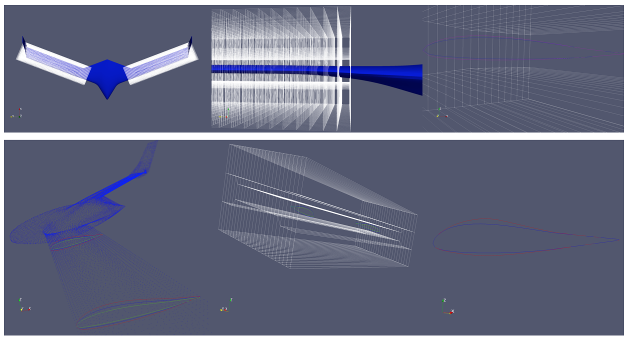

Figure 2: $\{x^4, \cdots, x^{33}\}$ modify the airfoil shape of the wing.

In order to compute the lower-dimensional subspace with the autoencoder, gradients of the objective function needs to be computed and used as training data. For the mAEWing2 problem, the gradient computed at point $\mathbf{x}_j$ is defined as:

$$
\nabla f(\mathbf{x}_j) = \nabla \left(\frac{L(\mathbf{x}_j)}{D(\mathbf{x}_j)}\right)^2 = \left[\frac{\partial \left(\frac{L(\mathbf{x}_j)}{D(\mathbf{x}_j)}\right)^2}{\partial x_j^1}, \cdots, \frac{\partial \left(\frac{L(\mathbf{x}_j)}{D(\mathbf{x}_j)}\right)^2}{\partial x_j^{33}}\right]^T \tag{5}
$$

where x_j^i is the component i of the vector $\mathbf{x}_j$. Next section describes how the dataset is generated.

3 Dataset construction

In order to build a dataset for training and testing the autoencoder, gradients of the objective function need to be computed. The data is generated using computer simulations. More specifically, for a given vector of parameters $\mathbf{x}_i$, the gradient $\nabla f(\mathbf{x}_i)$ can be computed analytically using a computer simulation (if the objective function has an analytical form, the gradients can also be computed directly). Let's assume N is the size of our dataset. Then, N points in the parameter space are sampled randomly using latin hypercube sampling: $\{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$. After, sampling these points, the gradients are computed at these randomly sampled points: $\{\nabla f(\mathbf{x}_1), \cdots, \nabla f(\mathbf{x}_N)\}$. Thus, the dataset has dimension of N units and contains N gradients vectors. Because computing N gradients using computer simulation is expensive for building the database, N has to be small. Therefore, because the dataset is not very large, 70% of the dataset is used for the training and 30% as dev/test set. The data is randomly shuffled before splitting the data into the different sets. Moreover, the autoencoder is an unsupervised application of neural networks and therefore it only uses inputs (the gradients computed).

3.0.1 Choosing the size of the dataset N

In order to choose the size of the dataset, an iterative process is used. An initial guess for N is the dimension of the parameter space, $N = N_x$. The dataset is generated and the data is splitted into training and dev set. Once the autoencoder is trained, the loss function (which will be described in a later section) is computed on the training and dev set. If the loss of the dev set is high compared to the loss of the training data, then more data is collected: $N_{new} = N + N_x$. The process is repeated until the loss function on both training and dev set gives similar values. This process of choosing N iteratively allows to have a small dataset and a model that does not have high variance. Moreover, this iterative process is feasible since training the autoencoder is not computationally expensive and it a has a small cost compared to the cost of generating the dataset using computer simulations. Next section describes different autoencoder architectures proposed for the task of computing the active subspace and the training strategy for these models.

4 Model description and training

4.1 Autoencoder architecture

The purpose of the autoencoder for this project is to find latent lower-dimensional state-space of the dataset of gradients pre-computed. For a given input vector $\nabla f(\mathbf{x}_i)$, the encoder computes a nonlinear mapping of the inputs as

$$
\mathbf{x}_{r_i} = h(\nabla f(\mathbf{x}_i), \mathbf{\Theta}_e) \tag{6}
$$

where $\mathbf{x}_{r_i}$ is the lower-dimensional representation of the input $\nabla f(\mathbf{x}_i)$ and $h(\cdot)$ is a mapping nonlinear function: $h : \mathbb{R}^{N_x} \rightarrow \mathbb{R}^{n_x}$. $\mathbf{\Theta}_e$ represent a set of parameters of the encoder. The encoded features, $\mathbf{x}_{r_i}$, are then decoded to reconstruct the given input vector $\nabla f(\mathbf{x}_i)$ using

$$
\widetilde{\nabla f(\mathbf{x}_i)} = g(\mathbf{x}_{r_i}, \mathbf{\Theta}_d) \tag{7}
$$

where $\widetilde{\nabla f(\mathbf{x}_i)}$ is the approximated reconstruction of the input $\nabla f(\mathbf{x}_i)$ and $g(\cdot)$ is a mapping nonlinear function: $g : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{N_x}$. $\mathbf{\Theta}_d$ represent a set of parameters of the decoder. Figure (3) shows the autoencoder architecture. Once the autoencoder is trained, the final purpose is using the encoded features, $\mathbf{x}_r$ as optimization variables for solving the optimization problem in the nonlinear active subspace.

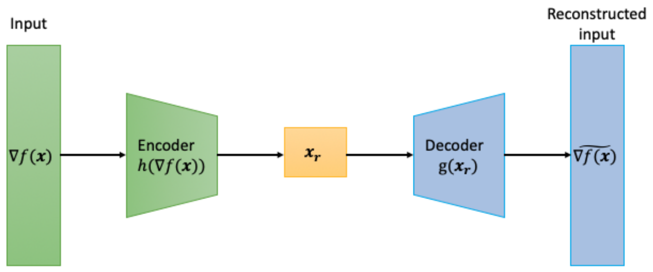
Input
$\nabla f(\boldsymbol{x})$
Encoder
$h(\nabla f(\boldsymbol{x}))$
$\boldsymbol{x_r}$
Decoder
$g(\boldsymbol{x_r})$
Reconstructed input
$\widehat{\nabla f(\boldsymbol{x})}$

Figure 3: Autoencoder for learning a low-dimensional representation of the gradient $\nabla f(\mathbf{x})$: $\nabla f(\mathbf{x}) \approx g(\mathbf{x}_r)$

Three different architecture are chosen and compared:

1. a fully-connected autoencoder which only consists of N_{FC} fully-connected layers in the encoder and N_{FC} fully-connected layers in the decoder
2. a convolutional autoencoder which only consists of N_{Conv} convolutional layers with 1D filters in the encoder and N_{Conv} transposed convolutional layers in the decoder
3. an autoencoder with N_{Conv} convolutional layers with 1D filters and one fully-connected layer in the encoder and one fully connected layer and N_{Conv} transposed convolutional layers in the decoder

The exponential linear unit (ELU) activation function is used for nonlinearity in all the architectures. The hyperparameters of the architecture, number of layers and number of hidden units for fully-connected layers or number of filters for convolutional layers are chosen using a grid search and evaluating the accuracy of the architecture on the dev set after being trained using the training set. Next subsection describes how accuracy is defined for the autoencoder.

4.2 Training strategy

In order to train the autoencoder, the training set is used. The reconstruction error (RE) for the single input $\nabla f(\mathbf{x}_i)$ is computed as:

$$RE(\nabla f(\mathbf{x}_i), \mathbf{\Theta}_e, \mathbf{\Theta}_d) = \sum_{j=1}^{N_x} \left(\frac{\partial f(\mathbf{x}_i)}{\partial x^j} - (g(\mathbf{x}_{r_i}(\mathbf{\Theta}_e)), \mathbf{\Theta}_d)^j \right)^2 \tag{8}$$

where $\mathbf{\Theta}_e$ and $\mathbf{\Theta}_d$ represent a set of parameters of the autoencoder, $g(\cdot)$ is the nonlinear decoder function and $g(\mathbf{x}_{r_i})^j$ represent the j-th component of the reconstructed vector. During the unsupervised training, the network tries to find the optimal values of the set of parameters $\mathbf{\Theta}_e$ and $\mathbf{\Theta}_d$ in order to minimize the reconstruction error for all the inputs:

$$Loss = \frac{1}{m} \sum_{i=1}^{m} RE(\nabla f(\mathbf{x}_i), \mathbf{\Theta}_e, \mathbf{\Theta}_d) \tag{9}$$

where m is the number of sample units. Using the loss function, the accuracy of the autoencoder can be estimated. The optimization is solved using the adaptive moment estimation (ADAM) algorithm with a learning rate $\alpha = 0.01$. This learning rate, optimal for this problem, is chosen using cross validation testing several values for the learning rate.

4.3 Model selection

For choosing the autoencoder to use among the three architectures proposed and the related hyperparameters associated with it, a grid search is used selecting the model and related hyperparameters with the smallest loss function evaluated using the dev set. Finally, once the architecture has been chosen, the loss of the chosen autoencoder is evaluated using the test set to evaluate the accuracy of the model on an unseen set of data and avoid overfitting. The hyperparameters to choose are:

- Number of layers: it ranges between 2 and 5
- For fully-connected layers, number of hidden units ranges between n_x and N_x
- For convolutional layers, number of filters in each layer ranges between n_x and N_x, where each filter is a 1D 3×1 filter.

where N_x is the dimension of the optimization parameter space and n_x is the dimension of the low-dimensional subspace. Moreover, the bottleneck layer, the layer which contain the low-dimensional representation of the input gradient has dimension n_x.

5 Results

In this paper the deep learning models are implemented in Python using Keras, which uses TensorFlow as its tensor flow manipulation. The deep learning models are trained using the training set with batch size equal to the number of training points (mini-batch is not used since the training set is not large). All the computations are run in a MacBook Pro with 2.5 GHz Quad-Core Intel Core i7 and memory 16 GB 1600 MHz DDR3.

5.1 Results for model selection

All the models are trained using the same training set and evaluated using the same dev set. For the purpose of selecting a model, the mAEWing2 optimization problem is used. The size of the database is $N = 500$. After running a grid search for selecting the model with the lowest dev set loss, the second architecture, the convolutional autoencoder, is selected. The lowest loss is equal to $Loss = 21.8$. The optimal architecture has 3 layers: the first layer has 33 filters, the second layer has 23 filters and the third layer has 3 filters (equal to n_x). For the first architecture, the autoencoder with fully connect layers, the lowest loss achieved is 36.6 using 3 layers with 33, 31 and 3 hidden units. For the third architecture, the autoencoder with convolution and fully-connected layers, the lowest loss achieved is 36.7 using 3 convolutional layers with the number of filters equal to 33, 33 and 33 and a fully-connected layer with 14 hidden units. Finally, using the chosen model, the loss for the test set is equal to 24.7. Thus, the model generalize with unseen data. In the process of choosing the best model, different activation functions are tested, such as RELU,Tanh and ELU: ELU is chosen because of the performances. Next section shows the autoencoder applied to solving optimization problems and its performances are compared with baseline methods.

5.2 Validation results

In order to validate the proposed methodology, different optimization problems are solved using three different methodologies:

1. using the full dimensional space:
   $\min_{\mathbf{x}} f(\mathbf{x})$ with optimization variables $\mathbf{x} \in \mathbb{R}^{N_x}$
2. using linear active subspace, $\mathbf{x} \approx \mathbf{U}_x \mathbf{x}_r$:
   $\min_{\mathbf{x}_r} f(\mathbf{U}_x \mathbf{x}_r)$ with optimization variables $\mathbf{x}_r \in \mathbb{R}^{n_x}$
3. using nonlinear active subspace, $\mathbf{x} \approx g(\mathbf{x}_r)$:
   $\min_{\mathbf{x}_r} f(g(\mathbf{x}_r))$ with optimization variables $\mathbf{x}_r \in \mathbb{R}^{n_x}$

where $N_x < n_x$. First, four unconstrained optimization problems are solved. Then, the mAEWing2 design optimization problem is solved.

5.2.1 Unconstrained optimization

The four unconstrained optimization problems used are benchmark problems used in large scale optimization described [2] and [3]:

- Rastrigin function
  (global minimum is $f(0, \cdots, 0) = 0$):

$$f(\mathbf{x}) = 10N_x + \sum_{i=1}^{N_x} \left(x_i^2 - 10cos(2\pi x_i)\right)$$

- Ackley function
  (global minimum is $f(0, \cdots, 0) = 0$):

$$f(\mathbf{x}) = -20e^{\left(-0.2\sqrt{\frac{1}{N_x}\sum_{i=1}^{N_x} x_i^2} - e^{\left(\frac{1}{N_x}\sum_{i=1}^{N_x} cos(2\pi x_i)\right)}\right)} + 20 + e$$

- Rosenbrock function
  (global minimum is $f(1, \cdots, 1) = 0$):

$$f(\mathbf{x}) = \sum_{i=1}^{N_x - 1} \left(100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2\right)$$

- Styblinski-Tang function
  (global minimum is
  $f(-2.903534, \cdots, -2.903534) = -39.16617N_x$):

$$f(\mathbf{x}) = \frac{\sum_{i=1}^{N_x} x_i^4 - 16x_i^2 + 5x_i}{2}$$

The optimization algorithm used to solve the problems is the genetic algorithm differential evolution. The maximum number of iteration is set to 500 and population size is 30. For all the optimization problems, the dimension of the optimization parameter space is $N_x = 33$. The dimension of the subspace is $n_x = 2$ for both linear and nonlinear active subspace.

Table 1: Summary for dataset construction and training

Function	Computing gradients [s]	Training, linear active subspace [s]	Training, nonlinear active subspace [s]
Rastrigin	0.284	0.004	3.018
Ackley	0.070	0.004	3.014
Rosenbrock	0.052	0.004	3.581
Styb.-Tang	0.193	0.004	3.143

Table 2: Summary for the optimization history

Function	Full space [s]	Found sol?	linear A.S. [s]	Found sol?	nonlinear A.S. [s]	Found sol?
Rastrigin	53.9	NO	0.5	YES	2.7	YES
Ackley	33.8	YES	0.4	YES	2.5	YES
Rosenbrock	31.6	YES	0.1	NO	1.8	YES
Styb.-Tang	26.3	YES	0.2	NO	1.5	YES

Dataset and training The number of gradients collected for training the autoencoder is N=50 for all the optimization problems. The gradients are computed at N randomly sampled points in the parameter space using latin hypercube sampling. The same dataset is used for computing the linear active subspace and nonlinear active subspace using the autoencoder. Table (1) shows a summary for the time spent in seconds for both constructing the database and training the models for linear active subspace and nonlinear active subspace. The training of the linear model includes only computing the SVD for the matrix of gradients. For training the autoencoder, The ADAM algorithm is used with a learning rate of 0.01, a validation split of 0.2 and early stopping is used to avoid overfitting. The table shows that the time for constructing the dataset and training is very small. This is because for all the optimization problems considered, the objective functions has an analytical form and therefore computing $N = 50$ gradients is not computationally expensive. Additionally, training both the linear and the nonlinear model is also not expensive since the size of the dataset is small.

Results Once all the models are trained, the optimization problems are solved using three methodologies: solving the problem using the full parameter space, solving with linear active subspace and nonlinear active subspace. Figure (4) shows the optimization history for the four optimization problems considered, where the horizontal axis represents the iteration and the vertical axis represents the objective function value. It can be observed that for all optimization problems, the methodology using nonlinear active subspace converges to the global optimal point and has the smallest number of iterations; moreover, both the methodology using the full space and the linear active subspace fail to find the global optimal point in at least one or more cases. Finally, table 2 shows the time spent for solving the optimization problems and whether the algorithm has found the solution of the problem. Both linear and nonlinear active subspace methodologies converge faster than the full space methodology.

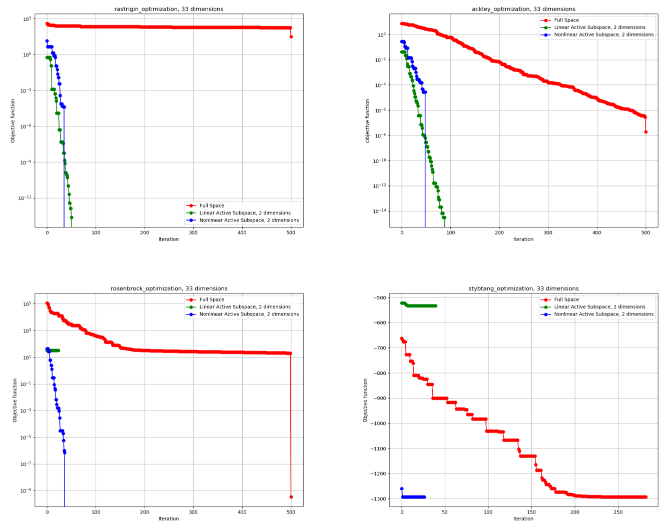

Figure 4:
Upper left: optimization of the **Rastrigin** function,
Upper right: optimization of the **Ackley** function,
Lower left: optimization of the **Rosenbrock** function,
Lower right: optimization of the **Styblinski-Tang** function.

Red curve: **full space**, green curve: **linear active subspace**, blue curve: **nonlinear active subspace**.

Table 3: Summary for dataset construction and training

Problem	Computing gradients [hours]	Training, nonlinear active subspace [hours]
mAEWing2	41.25	0.0023

5.2.2 mAEWing2 design optimization problem

The proposed methodology is also applied to a constrained optimization problem described in eq. 4: the design optimization of mAEWing2 aircraft. The optimization algorithm used to solve the problems is the Sequential Least Squares Programming (SLSQP). The maximum number of iteration is set to 500. The dimension of the optimization parameter space is $N_x = 33$. The dimension of the subspace is $n_x = 5$ for the nonlinear active subspace. In order to evaluate the objective function and its gradient for the mAEWIng2 problem, a computer simulation is run; each computer simulation is run in parallel using 6 nodes with 16 cores per node for a total of 96 cores. For this problem the linear active subspace methodology is not reported because lack of time. In the coming weeks, the author also plan to do a comparison between the performances of linear and nonlinear active subspace for the mAEWing2 problem.

Dataset and training The number of gradients used for training the autoencoder is N=50. The gradients are computed at N randomly sampled points in the parameter space using latin hypercube sampling. Table (3) shows a summary for the time spent in hours for both constructing the database and training the model for nonlinear active subspace. As for the previous problems, for training the autoencoder, the ADAM algorithm is used with a learning rate of 0.01, a validation split of 0.2 and early stopping is used to avoid overfitting. The table shows that the time for constructing the dataset is much bigger than the time spent for training. This is because evaluating the gradient requires running an expensive computer simulations while training is not expensive since the dataset is small.

Results The optimization problem is solved using two methodologies: using the full parameter space and nonlinear active subspace. Figure (5) shows the optimization history, where the horizontal axis represents the iteration and the vertical axis represents the objective function value. It can be observed that the methodology using nonlinear active subspace converges to a local optimal point and has a smaller number of iterations. The methodology using full space converges to a slightly better point. In order to find the global optimal solution of the problem, a global algorithm such basin-hopping algorithm has to be used which requires more iterations and the author plans to do it in the coming weeks. The methodology using the full space solved the problem in 27.37 hours while the one with the autoencoder took 20.52 hours. The methodology with the nonlinear active subspace converged faster, however it spent 41.25 hours for creating the dataset for the training. Thus, using a dimensionality reduction technique, such as linear active subspace or nonlinear active subspace, pays out if a global optimization algorithm is used and many more iterations and function queries are needed (as in the previous section for the unconstrained optimization examples). In fact, the initial cost of training and building the dataset needs to be a small percentage of the total time of the optimization problem in order to be computationally feasible.

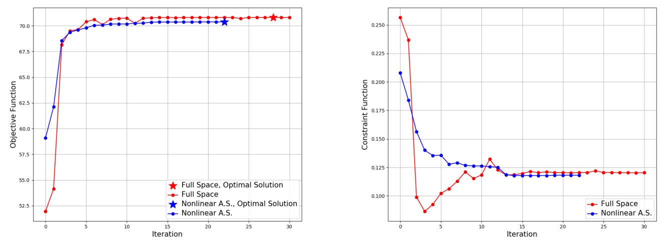

Figure 5: Left: Objective function history, Right: Constraint function history. Red curve: **full space** and blue curve: **nonlinear active subspace**.

6 Conclusion/Future Work

A novel methodology for dimensionality reduction is presented for solving highly parameterized optimization problems in a subspace using an autoencoder. Different methodologies are compared for solving different optimization problems. The initial results shows that using a nonlinear manifold for representing a subspace has advantages especially when a linear active subspace fails to capture nonlinearities in the problem. The author plan to study how to make the training of the autoencoder cheaper and to study the performances of this methodology in other additional problems.

7 Acknowledgments

The author would like to thank the CS230 teaching team for a great quarter and for teaching many useful concepts related to Neural networks which can be used for research and work related tasks.

8 Reference

1 Active subspace methods in theory and practice: Applications to kriging surfaces. Constantine, Dow and Wang.

2 Benchmark Functions for the CEC'2013 Special Session and Competition on Large-Scale Global Optimization. Li, Tang, Omidvar, Yang and Qin.

3 https://en.wikipedia.org/wiki/Test_functions_for_optimization