Universitat Politècninca de Catalunya
Computational Intelligence

Lab 2 - Evolutionary Algorithms

Roberto Bada, Abel Briones

30th December 2023

Contents

Abstract

In this study, we delve into the performance impact of using diverse Evolutionary Computation techniques as optimization methods for Multi-Layer Perceptrons. We aim to compare the efficacy of Genetic Algorithms and Evolutionary Strategies against the conventional derivative-based method of Stochastic Gradient Descent. Our investigation focuses on the interaction between these optimization techniques and the training dynamics of Artificial Neural Networks, providing insights into their relative strengths and limitations in the realm of supervised learning algorithms.

1 Introduction

Evolutionary Computation (EC) embodies a paradigm inspired by natural selection and genetic variation, fundamental principles of Evolutionary Biology. Its distinct role within the broader realms of Machine Learning and Artificial Intelligence is underscored by its capacity to provide versatile solutions, particularly valuable for uncharted or unresolved challenges. Central to this approach are Evolutionary Algorithms (EAs), a category of population-based, stochastic search algorithms. These algorithms, such as Genetic Algorithms (GAs) and Evolution Strategies (ESs), draw inspiration from the processes of natural evolution, leveraging the dynamics of mutation, selection, and crossover to navigate complex search spaces, and thereby evolving optimal solutions in an iterative manner.

Notably, EAs demonstrate a unique resilience against local optima and are less sensitive to initial conditions compared to derivative-based methods, though they may require longer computation times. Their robustness and global search capabilities make them an appropriate for complex optimization problems where traditional approaches may falter. This resilience is particularly beneficial in the realm of neural network optimization, where the landscape of possible solutions can be vast and intricate. By employing a population-based approach, EAs effectively explore a broader range of potential solutions, enhancing the likelihood of discovering superior configurations that might be overlooked by methods relying solely on gradient descent.

Their practical application as optimization procedures in ANNs is well-documented across various experimental studies. For instance, the implementation of GAs in a multi-layer Feedforward Neural Network (FNN) classifier, as explored in [1], yielded promising results in a complex classification problem. Similarly, Volna's investigation into the evolution of connection weights, architectures, and learning rules in neural networks illuminated the strategic advantage of incorporating evolutionary techniques at selective levels of the network design, as detailed in [2].

In the realm of Convolutional Neural Networks (CNNs), the study by Esfahanian and Akhavan [3] highlighted the efficacy of GAs, particularly in the early training phases, compared to conventional backpropagation methods. The utility of GAs extends to hyper-parameter optimization as well, as evidenced in studies like [4] and [5]. Beyond neural network optimization, the versatility of EC techniques is further exemplified in their application across diverse AI fields, including physics, as seen in [5] and [6]. Collectively, these studies underscore the potential of EC in enhancing the performance and efficiency of ANN optimization processes.

2 Problem Definition

The purpose of this experimental work is to explore the application of EC in the training of ANNs, with a particular focus on regression problems formulated using synthetic data. Our primary objective is to assess and compare the performance of a supervised Multi-layer Perceptron (MLP) learning algorithm when optimized through two evolutionary approaches: GA and ES. These methods will be evaluated against the conventional, derivative-based SGD.

Overall, this comparative analysis is designed to showcase the distinct strengths and limitations inherent to each method, offering insight into their efficacy within controlled experimental setting. By creating a synthetic data environment, we aim to ensure precise measurement and objective evaluation of the results, thereby providing a clear understanding of the potential and practicality of EC algorithms in neural network optimization.

3 Methodology

In Section 3.1, we explain the process followed when generating the data. Next, we define in Section 3.2 all the possible combinations in which data is generated for the benchmarking. In Section 3.3 we present the decision-making process concerning the implementation of both the MLP and its evolutionary optimization techniques, as well as other aspects of the problem setup, such as the performance metrics. Section 4 is devoted to the presentation of the results given by the different techniques. Here, we will also compare the three approaches. Finally, in Section 5, we present the conclusions of this exercise.

3.1 Data Generation

The synthetic data generated for this study is designed to encapsulate a wide range of scenarios, varying in complexity, noise levels, and dimensionality. We understand that ensuring diversity in data conditions is critical for providing a comprehensive evaluation platform. Thus, by encompassing scenarios from the simpler end of the spectrum to those with higher complexity and noise, we aim to gain insights into the performance dynamics of EAs. Specifically, we are interested in identifying the scenarios where EAs may exhibit superior performance or distinct advantages over conventional derivative-based methods.

Regarding our implementation, the synthetic data is generated using polynomial or sinusoidal functions, with controlled noise integration to mirror real-world data complexities. We modulate the complexity of these functions through hyper-parameters, tailoring the degree of the polynomial or the proportionality factor of the sinusoidal function. This allows us to create a variety of data scenarios, from simple to highly intricate patterns, facilitating a thorough evaluation of the optimization methods.

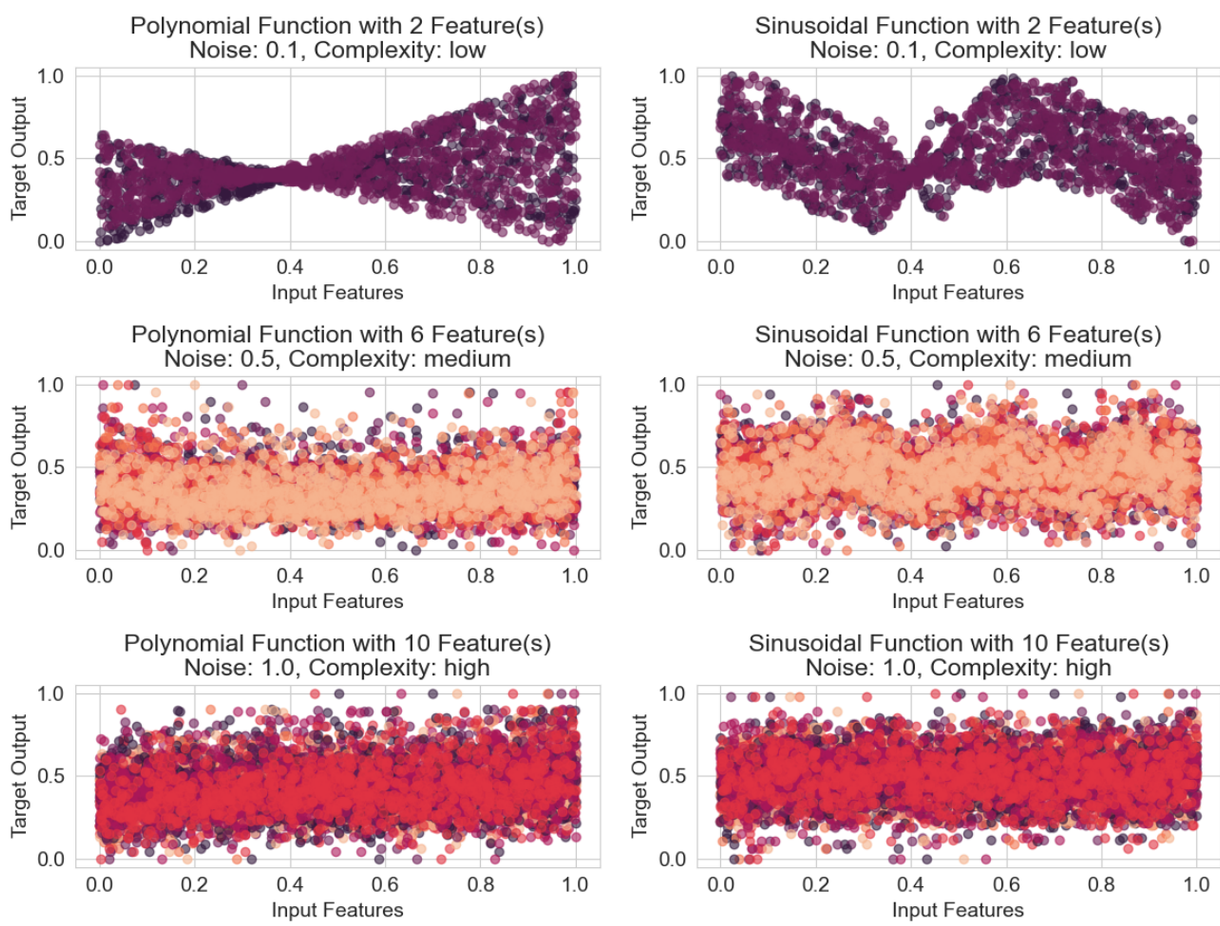

Figure 1: Example of polynomial and sinusoidal data with different levels of complexity and noise.

3.2 Experimental Setup

Our experimental setup encompasses various conditions to reflect different data characteristics. We explore feature dimensions of 3, 6, and 20, mirroring low, medium, and high data dimensionality. Additionally, noise levels are adjusted between 0.1, 0.5, and 1.0, offering a range of scenarios from subtle to significant data imperfections.

The adjustment of complexity levels from 'low' to 'high' further diversifies our test cases, ensuring a holistic assessment of each optimization method's performance under varying conditions. The noise level's modulation allows us to evaluate how different optimization methods respond to varying degrees of data uncertainty. This simulation of real-world data imperfections aims to provide insight into the robustness and adaptability of each algorithm.

In addition to the above considerations, our data preparation includes a crucial step of normalizing the target variable. Normalization aids in stabilizing the training process and enhances the convergence speed of the optimization algorithms, especially when dealing with data that have varying scales and distributions. Such preprocessing not only contributes to the robustness of the model but also ensures a fair comparison across different optimization methods by providing them with uniformly scaled data.

Lastly, the dataset is systematically divided into training, validation, and test segments, following a 70%, 15%, and 15% split, respectively. This distribution is meticulously adhered to in all experiments to ensure consistent and reliable model evaluation.

3.3 Multi-Layer Perceptron

The core of our experimental framework is a Multi-Layer Perceptron (MLP) with a single hidden layer. This architecture, characterized by its simplicity and adaptability, is instrumental in evaluating the impact of various training algorithms on the neural network's learning process. The MLP consists of an input layer, which matches the dataset's feature count, a hidden layer with an adaptable number of neurons, and an output layer tailored for regression tasks. The selection of this specific architecture is grounded in its broad applicability and ease of interpretation in diverse experimental settings.

For the activation function, the ReLU function was chosen due to its non-linear characteristics and computational efficiency. This function is notably effective in mitigating the issue of vanishing gradients during training, a significant advantage when working with evolutionary optimization methods that rely on the propagation and enhancement of favorable traits. To strike a balance between model complexity and computational efficiency, the number of neurons in the hidden layer is capped at 32. This constraint ensures the MLP is sufficiently complex to discern non-linear patterns without imposing excessive computational demands, thus facilitating rapid experimentation across various optimization setups. For derivative-based optimization experiments, the MLP was configured with 16 hidden neurons to align with the median value used in evolutionary technique studies.

The MLP's design is modular, enhancing its compatibility with different optimizers and allowing for the architectural evolution of the network. This flexibility has been achieved by implementing a class structure that supports dynamic adjustments in response to the optimizer's requirements. The MLP class includes methods for weight initialization and dynamic setting of network parameters. The weights of the hidden layer are initialized using the Kaiming uniform method, appropriate for ReLU activations, while the output layer weights follow the Xavier uniform initialization. This choice of initialization methods contributes to the network's effective training and convergence.

Moreover, the MLP class's capability to dynamically adjust its structure, such as modifying the hidden layer size and setting weights, aligns seamlessly with the EAs' needs. Familiarization with the Distributed Evolutionary Algorithms in Python (DEAP) package was achieved through diligent engagement with the best practices and comprehensive documentation [7], ensuring the effective implementation of EAs.

3.4 Evolutionary Techniques

As previously mentioned, apart from Stochastic Gradient Descent (SGD), the two primary optimization techniques employed for training the MLP are GAs and ESs. The parameters for these evolutionary methods were carefully chosen based on established practices in academic literature and documentation [8], forming a credible foundation for their "vanilla" configurations. For example, decisions regarding population sizes and mutation rates adhere to common practices in evolutionary computation, enabling extensive exploration of the solution space and maintaining diversity without excessive randomization [9].

Genetic Algorithms

A fixed set of parameters was utilized for the GA optimizer to manage the scope of experiments and focus on comparing different setups. The GA optimization process involves:

1. Initial population creation with random individuals.
2. Evaluation and selection of parents based on validation loss.
3. Crossover (80 % probability) to generate new individuals.
4. Mutation of offspring (10% probability), a choice based on literature recommendations [8].
5. Formation of the new population from these children.
6. Repeating steps 2-5 until reaching the maximum iteration count.
7. Selection of the best-performing model.

The initial population size was set at 100, and the number of generations at 10, balancing search space exploration with computational efficiency.

Evolutionary Strategies

The ES optimizer aims to find an optimal MLP configuration through a systematic process:

1. Generation of an initial population comprising random individuals.
2. Fitness evaluation using validation loss and selection of parents.
3. Offspring creation from selected parents.
4. Mutation of individuals at a 10% probability rate.
5. New population creation from these offspring.
6. Iterative execution of steps 2-5 until the maximum number of iterations is reached.
7. Identification and selection of the best model.

A fixed optimizer configuration was chosen to limit the experimental scope, with both the initial population and the number of generations set at 100 and 10, respectively.

Stochastic Gradient Descent

In contrast to the evolutionary techniques, the SGD optimizer follows a gradient descent approach with specific parameters:

1. Model optimization over a defined number of epochs with batch processing.
2. Use of Mean Squared Error (MSE) loss function.
3. Application of weight decay for regularization.
4. Iterative model training and evaluation using training and validation data.

The SGD optimizer, implemented in Python, utilizes PyTorch's capabilities for model training and evaluation. Key parameters include a learning rate of 0.01, a batch size of 32, and weight decay set at 0.01 for regularization purposes.

3.5 Performance metrics

The primary performance metrics used were the validation and test losses, which provide a straightforward measure of model accuracy on unseen data. The generalization error, which quantifies how well the model performs on new data, is inferred to be closely related to these losses. A model with a low loss is expected to have a lower generalization error, indicating it can effectively generalize from the training data to unseen data.

Alongside these key metrics, additional factors such as training time and the number of neurons in the hidden layer were also evaluated. While the latter is basically a measure of the complexity of the obtained model, the former gives us a measure of the efficiency of each algorithm, which translates into important insights when comparing two algorithms that are close in generalization error. Execution time was measured using Python's time module, providing an objective measure of the computational efficiency of each optimization method.

To ensure comprehensive and robust benchmarking, various conditions, including feature dimensions, noise levels, and complexity levels, were methodically tested. This approach facilitated a thorough evaluation of each optimization technique under diverse and challenging scenarios, mirroring real-world complexities. The benchmarking process, as scripted in main.py, involved generating synthetic data, splitting it into training, validation, and testing sets, and systematically applying each optimizer to the data. The results were then aggregated and analyzed to discern patterns and insights regarding the performance of each optimizer under different conditions.

Observation on computational resources: Just as a final note to this section, the computational resources available (32GB RAM, Intel Core i7-11800H CPU, 1TB storage) shaped the decision to use certain optimization techniques and parameters. The goal was to find a balance between model performance and resource constraints, ensuring that the benchmarking was feasible within the project timeline.

4 Results

The optimization performance of SGD, GA, and ES was assessed under the specified range of conditions. Referencing Table 1 and Table 2, SGD consistently outperformed GA and ES in terms of lower validation and test losses across varying dimensions, complexity levels, and noise levels. Notably, Table 2 details SGD's dominance even under higher complexity and noise conditions, which were theoretically favorable to evolutionary techniques.

4.1 Performance Evaluation

4.1.1 Validation and Test Losses

As shown in Table 2, SGD achieved the lowest validation loss across all tested scenarios, contradicting the expected advantage of GA and ES in more complex and noisy environments. This suggests that the fine-tuning capability of SGD is robust against a variety of challenges posed in the synthetic data.

4.1.2 Training Time

The comparative analysis of training times, visualized in the composite graph (see Figure 2), highlights the stark contrast between SGD's efficiency and the more time-consuming nature of the evolutionary approaches. Despite increasing feature dimensions and noise levels, SGD's training time remained relatively stable, whereas GA and ES exhibited significant increases.

4.1.3 Model Complexity

SGD maintained a consistent model complexity, as indicated by the number of neurons in the hidden layer, regardless of the varying conditions. This observation, referenced in Table 1, underscores the algorithm's capacity to achieve high performance without necessitating complex models.

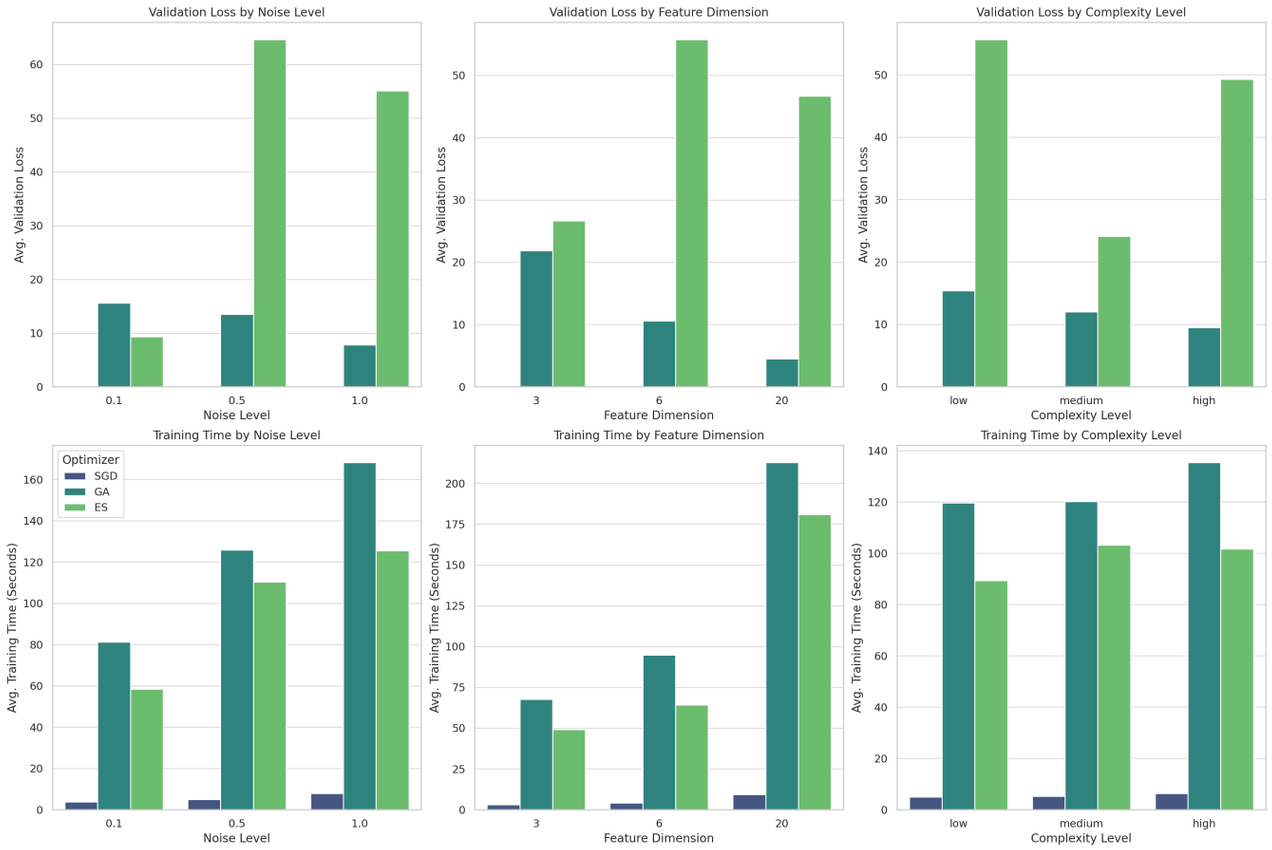

Figure 2: Average validation loss and training time across different noise levels, feature dimensions, and complexity levels for each optimizer.

4.1.4 Generalization Error

The generalization error, inferred from the gap between training and validation/test loss, was consistently lower for SGD. This suggests that SGD models were better at generalizing from training to unseen data, a key aspect of model robustness.

5 Conclusions

Our evaluation critically examined the optimization capacity of GA and ES over SGD. Contrary to expectations, SGD prevailed over EAs in all tested scenarios, raising questions about the experimental conditions favoring the theoretical strengths of EAs.

The results underscore some of the strengths and limitations inherent in both EC techniques and SGD. SGD's consistent outperformance in terms of loss metrics and training time underlines its efficiency and precision, particularly in less complex and lower noise environments. This reaffirms the suitability of gradient-based methods for problems where the solution space is well-behaved and gradients provide reliable guidance. Despite GA and ES not outperforming SGD in terms of accuracy and efficiency, their ability to explore diverse solutions was evident. This is particularly notable in higher noise settings where the robustness of EC can be advantageous. However, their inherent exploration trait comes at the cost of longer training times and potentially less precision, as was observed.

Despite several attempts to manually tune the hyperparameters of the EAs, we did not observe significant changes in the outcomes. Perhaps a more systematic and informed approach to parameter optimization could have been employed. Finally, future steps should also aim to refine the conditions under which EAs are benchmarked. A more nuanced approach to generating test scenarios that align more closely with the theoretical strengths of EAs could provide a fairer assessment of their performance. By enhancing the design of the benchmarking framework and expanding computational capabilities, we can pave the way for a more equitable comparison and a deeper understanding of the situational merits of EAs.

Acknowledgement of Computational Resources: The computational resources available significantly influenced the choice of optimization techniques and parameters, aiming to strike a balance between model performance and practical feasibility within the project's constraints.

References

[1] David J. Montana and Lawrence Davis. "Training Feedforward Neural Networks Using Genetic Algorithms". In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'89. Detroit, Michigan: Morgan Kaufmann Publishers Inc., 1989, pp. 762–767.

[2] Eva Volna. *Neuroevolutionary optimization*. 2010. arXiv: 1004.3557 [cs.NE].

[3] Parsa Esfahanian and Mohammad Akhavan. *GACNN: Training Deep Convolutional Neural Networks with Genetic Algorithm*. 2019. arXiv: 1909.13354 [cs.NE].

[4] Keshav Ganapathy. *A Study of Genetic Algorithms for Hyperparameter Optimization of Neural Networks in Machine Translation*. 2020. arXiv: 2009.08928 [cs.NE].

[5] Dwyer S. Deighan et al. "Genetic-algorithm-optimized neural networks for gravitational wave classification". In: *Neural Computing and Applications* 33.20 (Apr. 2021), pp. 13859–13883. ISSN: 1433-3058. DOI: 10.1007/s00521-021-06024-4. URL: http://dx.doi.org/10.1007/s00521-021-06024-4.

[6] Isidro Gómez-Vargas, Joshua Briones Andrade, and J. Alberto Vázquez. "Neural networks optimized by genetic algorithms in cosmology". In: *Physical Review D* 107.4 (Feb. 2023). ISSN: 2470-0029. DOI: 10.1103/physrevd.107.043509. URL: http://dx.doi.org/10.1103/PhysRevD.107.043509.

[7] Félix-Antoine Fortin et al. "DEAP: Evolutionary Algorithms Made Easy". In: *Journal of Machine Learning Research* 13 (July 2012), pp. 2171–2175.

[8] T. Back, U. Hammel, and H.-P. Schwefel. "Evolutionary computation: comments on the history and current state". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 3–17. DOI: 10.1109/4235.585888.

[9] Thomas Bäck. "Optimal Mutation Rates in Genetic Search". In: *Proceedings of the 5th International Conference on Genetic Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993, pp. 2–8. ISBN: 1558602992.

6 Annex

Benchmark Results Overview

Table 1: Best Algorithm Result

Optimizer	Dimensions	Complexity Level	Noise Level	Hidden Neurons	Training Loss	Validation Loss	Test Loss	Training Time
SGD	3	medium	0.1	16	0.000913	0.000766	0.000957	2.749002
GA	3	high	0.5	11	0.055985	0.052035	0.053853	61.926079
ES	3	medium	0.5	11	0.292887	0.309985	0.290232	44.160511

Table 2: Best Optimizer By Condition

Optimizer	Dimension	Complexity Level	Noise Level	Hidden Neurons	Training Loss	Validation Loss	Test Loss	Training Time
SGD	3	low	0.1	16	0.000976	0.000872	0.000905	2.782575
SGD	3	low	0.5	16	0.001384	0.001155	0.001230	2.785749
SGD	3	low	1.0	16	0.002354	0.002153	0.002240	2.766225
SGD	6	low	0.1	16	0.002406	0.002409	0.002630	4.148098
SGD	6	low	0.5	16	0.003036	0.003061	0.003659	4.168957
SGD	6	low	1.0	16	0.003736	0.003760	0.005467	3.896019
SGD	20	low	0.1	16	0.010511	0.011072	0.013644	4.178706
SGD	20	low	0.5	16	0.012822	0.013521	0.016172	4.125845
SGD	20	low	1.0	16	0.014106	0.014851	0.018354	15.514159
SGD	3	medium	0.1	16	0.000913	0.000766	0.000957	2.749002
SGD	3	medium	0.5	16	0.001322	0.001250	0.001583	2.758950
SGD	3	medium	1.0	16	0.002226	0.002143	0.002311	4.443471
SGD	6	medium	0.1	16	0.002769	0.003519	0.003024	4.517666
SGD	6	medium	0.5	16	0.002016	0.002335	0.002169	4.009094
SGD	6	medium	1.0	16	0.003068	0.003194	0.003151	4.256225
SGD	20	medium	0.1	16	0.014599	0.015531	0.015303	4.309628
SGD	20	medium	0.5	16	0.015349	0.015337	0.014900	4.044248
SGD	20	medium	1.0	16	0.012982	0.013410	0.013785	15.429142
SGD	3	high	0.1	16	0.002396	0.002319	0.002632	2.773666
SGD	3	high	0.5	16	0.002712	0.002759	0.003073	2.781287
SGD	3	high	1.0	16	0.003500	0.003383	0.004224	4.279972
SGD	6	high	0.1	16	0.003691	0.003830	0.004220	4.228162
SGD	6	high	0.5	16	0.005339	0.005212	0.005884	4.363696
SGD	6	high	1.0	16	0.005209	0.004919	0.005203	3.923979
SGD	20	high	0.1	16	0.007204	0.008561	0.006600	4.050054
SGD	20	high	0.5	16	0.007179	0.009626	0.006860	15.380072
SGD	20	high	1.0	16	0.007188	0.008881	0.006486	15.658854