Exploring the Capabilities of Zeroth-Order Methods in ML tasks

Anoush Azar-Pey
anoush.azar-pey@epfl.ch

Clémetine Jordan
clementine.jordan@epfl.ch

Damien Genoud
damien.genoud@epfl.ch

June 2025

1 Introduction

Optimization algorithms are essential for training machine learning models. *Gradient-based methods* leverage calculus to compute gradients (derivatives) of the loss function, guiding iterative weight updates toward minima. They excel in efficiency for differentiable models (like neural networks) but require smooth, calculable gradients. Conversely, *zeroth-order methods* [NS17] optimize without derivatives, relying solely on function evaluations. These are robust for non-differentiable, noisy, or black-box objectives, though typically slower and less precise. While gradient-based approaches dominate modern deep learning, zeroth-order techniques remain valuable for complex optimization landscapes where gradients are impractical or unavailable. In this project, we will investigate the performance of two different zeroth-order gradient estimations methods, SPSA and MPGE, on multiple machine learning problems.

2 Theory

We introduce multiple zeroth-order gradient estimation methods to be used alongside gradient descent when applicable.

2.1 Multi-point Gradient Estimation

2.1.1 2-point Estimator

First, we present *2-point Estimators*. They estimate the first-order gradient of a function by using the following formula:

$$\begin{aligned} &\text{Sample } u \text{ from } \{u : \|u\|_2 = 1\} \\ &\hat{\nabla}_k h(x) := \frac{h(x+uk) - h(x-uk)}{2k} \cdot u \qquad (1) \\ &x_{t+1} \leftarrow x_t - \eta \hat{\nabla}_k h(x_t) \end{aligned}$$

However, this approach may exhibit suboptimal performance in higher-dimensional problems, where the infinite-dimensional nature of the search space renders directional exploration insufficient for accurate gradient estimation.

2.1.2 $2d$-point Estimator

To address this limitation, we propose a $2d$-point estimator specifically designed for problems in $\mathbb{R}^d$. This method leverages orthogonal perturbations to span the full d-dimensional space, ensuring robust gradient estimation.

1. Sample a random perturbation direction δ_1 uniformly on the unit circle.
2. For each remaining dimension $i \in [d]$: Generate a new perturbation δ_i orthogonal to $\delta_1, \ldots, \delta_{i-1}$
3. Evaluate the objective function at m perturbed points: $x \pm \delta_i, \forall i \in [d]$ to compute each (1) estimator, and finally output their average per dimension.

We will refer to this algorithm as the `Multi-point Gradient Estimator` (MPGE).

2.2 SPSA

The `Simultaneous Perturbation Stochastic Approximation` (SPSA) [Wan20] estimates gradients using two function evaluations. For parameters $x \in \mathbb{R}^d$ and perturbation size K, the gradient approximation is:

$$\hat{\nabla} f(x) = \Delta \cdot \frac{f(x + K\Delta) - f(x - K\Delta)}{2K} \qquad (2)$$

where $\Delta \in \mathbb{R}^d : \Delta_i = \pm 1$ with equal probability.

3 Experiments

To compare and explore these methods, we now implement them on various problems, which we introduce to then present and explain their results.

3.1 Linear regression

3.1.1 2D problem

We designed two simple linear regression problems to compare them with a typical gradient-based method, that we will refer to as `Analytical`. The first, a two dimensions problem, with one bias and one weight decision parameter. We use the MSE loss as our loss function and a learning rate of 0.1. We generate 1000 data samples using the following formula:

$$X \sim \text{Uniform}(0, 1), \quad \epsilon \sim \mathcal{N}(0, 0.4) \qquad (3)$$

$$y = 2X + 3 + \epsilon. \qquad (4)$$

Algorithm	Loss	Parameter error
SPSA	0.156	0.0934
MPGE	0.156	0.0946
Analytical	0.156	0.0934

Table 1: Algorithm performance: average of the last 10 iterations

For each algorithm, we display in Figure 3, on the left, the loss function over iterations and on the right, the $L2$ distance between our decision variables and the actual weight and bias used for data generation. In Table 3.1.1 we report the average of same two metrics for the last ten iterations of these algorithms. Finally, in Figure 2 we plot the generated data, the true y line and our estimated y line using SPSA.

We can see that both MPGE and the `Analytical` method converge similarly, with the former taking more iterations. In constrast, SPSA first found a solution that minimizes the parameter error but not the MSE loss; it then converged to the same solution as the other two.

MPGE exhibits significantly higher computational complexity and running-time than both SPSA and `Analytical` gradient descent, requiring $\mathcal{O}(d)$ objective function evaluations per iteration.

3.1.2 11D problem with 5 sparse dimensions

Similarly, we present a new linear regression problem with 10 dimensional data and learnable weights and 1 bias. We use the same loss function and learning rate. To increase problem complexity, we randomly sparsify 5 of the 10 feature dimensions by setting both the data values and true model coefficients to zero in these dimensions. Again, we generate 1000 data samples using the following formula:

$$X \sim \mathcal{N}(0, I_{10}), \quad y = 2 \sum_{j \in \mathcal{J}} x_j + 3 \tag{5}$$

where $\mathcal{J}$ denotes the set of non-sparse feature indices, and x_j represents the j-th feature component.

Algorithm	Loss	Parameter error
SPSA	0.1597	9.6883
MPGE	0.1597	0.0703
Analytical	0.1597	0.0701

Table 2: Algorithm performance: average of the last 10 iterations

Again, MPGE and the `Analytical` method converge similarly but this times SPSA is the slowest to converge. The first two managed to find a solution with a small parameter error, while the third got stuck with a higher value. It might be due with how these algorithms handled the sparse dimensions. Again all three algorithms converged ot the same solution.

3.2 Soft K-Means

3.2.1 2D clustering problem

To evaluate our optimization methods on clustering problems, we implement the Soft K-Means (Fuzzy C-Means) algorithm. We generate 300 synthetic data samples with 5 clusters in 2-dimensional space using isotropic Gaussian blobs ($\sigma = 5$). The objective function minimizes the fuzzy within-cluster sum of squares:

$$\mathcal{J}(C) = \sum_{i=1}^{n} \sum_{j=1}^{c} w_{ij}^{m} \|x_i - c_j\|^2 \tag{6}$$

where $C = \{c_j\}_{j=1}^{c}$ are cluster centers, $m = 2$ is the fuzziness parameter and w_{ij} is the fuzzy membership weight, computed via:

$$w_{ij} = \left[\sum_{k=1}^{c} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{2/(m-1)} \right]^{-1}. \tag{7}$$

We initialize cluster centers near random data points: $C_{\text{init}} = X_{\text{random}} + \mathcal{N}(0, 0.1)$ and pick 10^{-3} as our learning rate.

Algorithm	Loss
SPSA	4101.9843
MPGE	4150.6167
Analytical	4100.4138

Table 3: Algorithm performance: average of the last 10 iterations

Figure 6 shows optimization progress, displaying objective loss over iterations. Table 3.2.1 reports final performance metrics. Figure 5 displays all generated plots with their attributed class and their cluster center. SPSA and the `Analytical` method converge to similar solutions. MPGE is slower to converge and its solution is slightly less optimal.

3.2.2 3D clustering problem with 3 sparse dimensions

We increase complexity by generating 3 clusters in 6-dimensional space with 3 sparse dimensions, zeroed in data samples.

Algorithm	Loss
SPSA	9805.3763
MPGE	9817.708
Analytical	9805.3763

Table 4: Algorithm performance: average of the last 10 iterations

We observe the same behavior and dynamics as in the last problem, where MPGE is slower and its solution slightly less optimal.

3.3 Reinforcement Learning

To test the performance of SPSA and MPGE on a more complicate problem, we apply them to the Mountain Car reinforcement learning task from Gymnasium. In this environment, a car must build momentum to reach the top of a hill. An image of the environment can be seen in Figure 8. The agent receives a reward at each timestep until it reaches the goal. The piecewise reward function (to maximize) is non-differentiable, making it a suitable candidate to optimize for zeroth-order methods like SPSA or MPGE.

3.3.1 Setting and Modifications

We use simple MLP with one hidden layer of dimension 8 and a `tanh` activation function. The input consists of the vector (position, velocity), and the output is one of three possible actions: accelerate to the left, do nothing, or accelerate to the right. The parameters of the network are optimized directly using SPSA or MPGE. This means that the problem is now in 51 dimensions.

To help with the optimization process (otherwise the algorithm never converges), we introduce a reward function that helps build momentum while penalizing not reaching the goal. After each action, the reward is computed as:

$$r = \mathbb{I}_{\text{goal}} \cdot 100 + 5 \cdot |\text{velocity}| + 0.2 \cdot (\sin(3 \cdot \text{position}) + 1) - 1, \tag{8}$$

where $\mathbb{I}_{\text{goal}}$ equals 1 if the car reaches the goal. Note that the network learns to maximize this quantity.

Moreover, only parameter updates that improve the average reward are applied to make the optimization more robust to bad updates. Without this modification, we found that both SPSA and MPGE fail to reach the goal a single time.

3.3.2 Results

At each time step, the model is evaluated by computing the average reward the car gets with the newly updated parameters over five episodes. Table 5 displays the best results of SPSA for some combinations of α and K obtained with a grid-search. A plot that highlights the rate of convergence of these combinations can be seen in Figure 9.

$\alpha \backslash K$	10.0	5.0	1.0
1.0	-151.64	11.48	-176.44
0.5	-183.40	**30.83**	-176.44
0.25	-182.46	-182.62	14.58

Table 5: Best mean rewards obtained by SPSA over five episodes during training for different combinations of α and K. Positive values indicate that the car reaches the goal consistently while large negative values indicate that the car never reaches it.

The results show that SPSA seems to be sensitive to the choice of α and K. Indeed, when K is too large, the gradient can become biased and small in magnitude while, conversely, small K values lead to large and noisy gradient estimates. Thus, in our experiments, we noticed that choosing $\alpha = K/10$ is a good rule-of-thumb for reasonable updates. We also note that the randomness of both the car's starting positions, especially during early training, and the SPSA updates play a major role in the results obtained: it is possible for SPSA to never converge by consistently making bad gradient estimations.

In contrast to SPSA, MPGE never manages to reach the goal, regardless of the hyperparameters. An example of training is shown in Figure 10. This is probably due to MPGE getting stuck in local optima: the lower level of randomness compared to SPSA makes it less capable of escaping them once reached. Additionally, the randomness of SPSA allows for better exploration of the parameter space, which is important in order reach the goal a first time. Finally, due to the number of dimensions being large ($D = 51$), an iteration of MPGE takes about 51 times longer than for SPSA, making the former ill-suited to train our model efficiently.

3.4 Architecture Optimization

Recent research, such as the framework ZARTS, uses zeroth-order methods to perform Neural Architecture Search (NAS) via Random Search (RS), Model-Guided Search (MGS), and Gradient-less Descent (GLD) [Pfe+23]. In our case, we decided to explore the ability of the SPSA algorithm for NAS, where we test its performance on a simple 3-layer MLP architecture, compared to a basic grid search.

3.4.1 Experiment Settings

The function we optimized is the validation loss of a trained model with respect to the number of units in each layer. Our SPSA gradient is calculated similarly to Equation 2, where f computes the validation loss of the model trained using the architecture mapped from the vector x.

At each iteration k, we update:

$$x_{k+1} = \text{clip}(x_k - a_k \cdot \hat{g}_k^{\text{avg}}, 0, 1) \tag{9}$$

where a_k is the learning rate and $\hat{g}_k^{\text{avg}}$ the smoothed gradient. We tested our optimization across several initial vectors and learning rate values, and compared the resulting architectures to those obtained via a simple grid search. All experiments were conducted on the small UCI Wine Quality dataset [Cor+09].

3.4.2 Results

Our results vary significantly across runs without a seed, indicating that SPSA may lack stability. The gradient estimates often oscillate without clear direction, and early convergence to a lucky validation loss due to noise can hinder performance. However, trends still emerged. The optimization is highly sensitive to the initial vector x. Starting from a small, suboptimal architecture often leads to fast increases in layer sizes and improved performance, as shown in Figure 11. In contrast, as in Figures 12 and 13, initializing with a bigger architecture causes the algorithm to remain in that region. Both produce results around 58–59% accuracy, similar to the best architecture found by grid search, while the worst configuration yields around 50%. SPSA tends to favor larger architectures and rarely reduces layer sizes, which could lead to overfitting. Also, as shown in Figure 14, starting small with a low learning rate step size achieves a smaller final architecture, but the influence on accuracy is marginal. While SPSA can quickly attain a satisfying architecture, its dependence on hyperparameter choices makes it less practical and generalizable. Finally, showing SPSA's full potential on NAS may require larger datasets or more complex architectures than the one considered here.

4 Conclusion

We compared the performance of SPSA and MPGE on different machine learning tasks, ranging from very simple to high-dimensional, non-differentiable, and black-box problems. We concluded that both methods perform comparably to gradient descent on simple ML tasks (Linear Regression, K-Means). We then showed that SPSA can also be used to solve more complex tasks (Reinforcement Learning, NAS), even though it can be impractical due to its strong dependence on initial conditions and hyperparameters. On the other hand, we found that MPGE is ill-suited for high-dimensional, non-differentiable problems: its time complexity is linear in the number of parameters, making it much slower than SPSA. Moreover, its lack of stochasticity often causes MPGE to get stuck in local minima. Overall, SPSA displays good convergence and exploratory behaviors across a wide range of problems, making it an appealing alternative to gradient descent in certain situations.

References

[Cor+09] Paulo Cortez et al. "Modeling wine preferences by data mining from physicochemical properties". In: *Decision Support Systems* 47.4 (2009), pp. 547–553.

[NS17] Yurii Nesterov and Vladimir Spokoiny. "Random Gradient-Free Minimization of Convex Functions". In: *Foundations of Computational Mathematics* 17.2 (2017), pp. 527–566. ISSN: 1615-3383. DOI: 10.1007/s10208-015-9296-2. URL: https://doi.org/10.1007/s10208-015-9296-2.

[Pfe+23] Jonas Pfeifer et al. "ZARTS: A Zero-order Architecture Search on Training and Search Instabilities". In: *International Conference on Learning Representations (ICLR)*. 2023.

[Wan20] Chen Wang. *An overview of SPSA: recent development and applications*. 2020. arXiv: 2012.06952 [math.OC]. URL: https://arxiv.org/abs/2012.06952.

A Additional figures

A.1 2.1.2

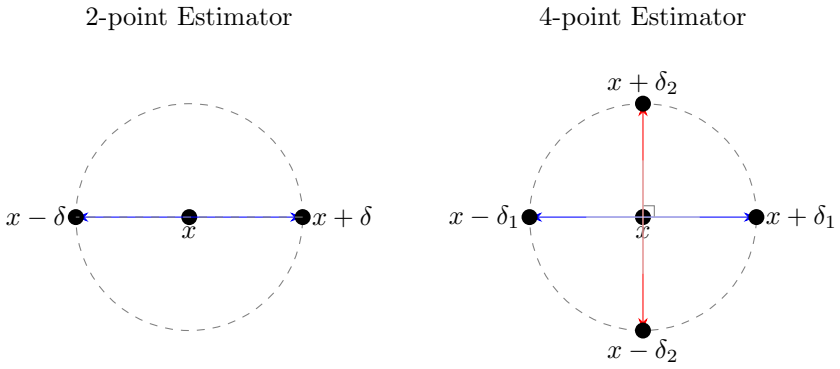

Figure 1: Gradient estimation methods in 2D space: Left: 2-point estimator with symmetric perturbations $\pm\delta$ in a single direction, spanning only a line. Right: 4-point estimator with orthogonal perturbations $\pm\delta_1$ and $\pm\delta_2$, enabling full 2D span through linear combinations.

A.2 3.1.1

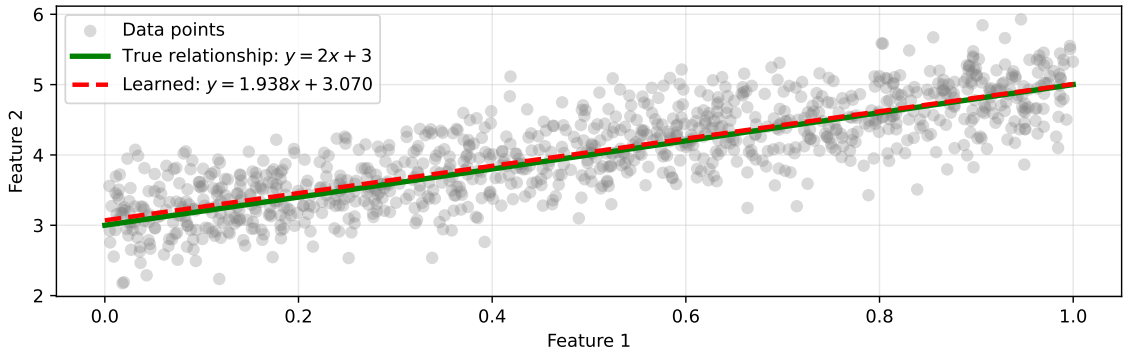

Figure 2: Generated data, true and learned relationship with SPSA

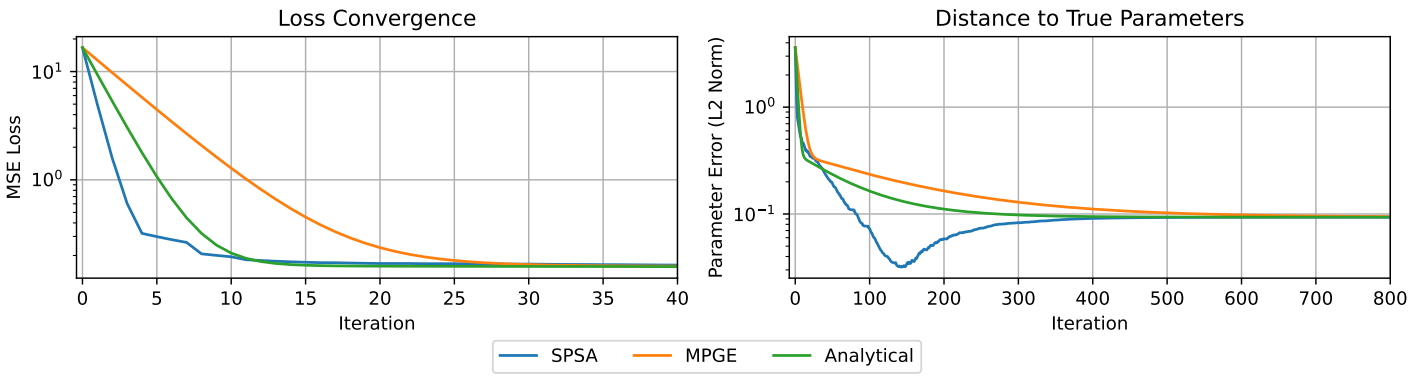

Figure 3: Convergence of SPSA, MPGE and the `Analytical` first-order gradients

A.3 3.1.2

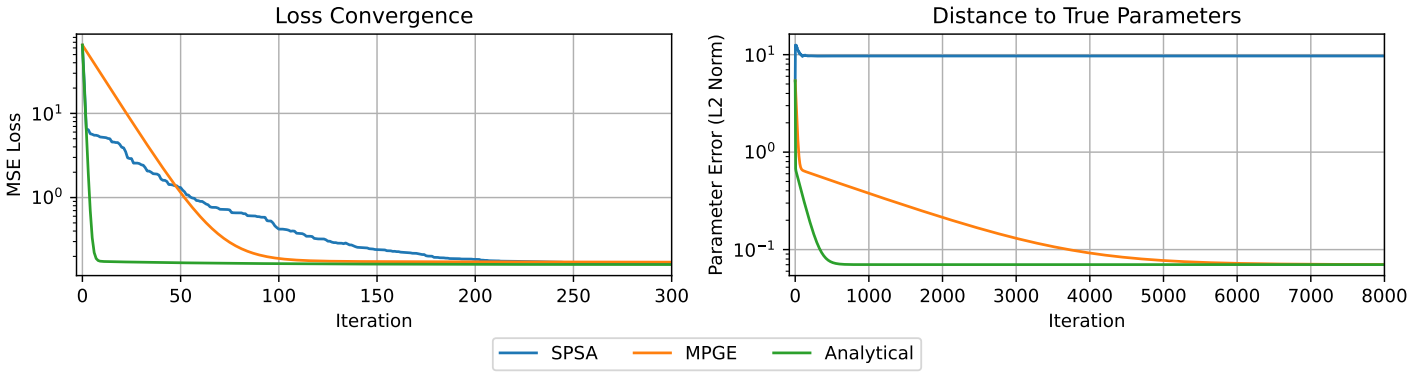

Figure 4: Convergence of SPSA, MPGE and the `Analytical` first-order gradients

A.4 3.2.1

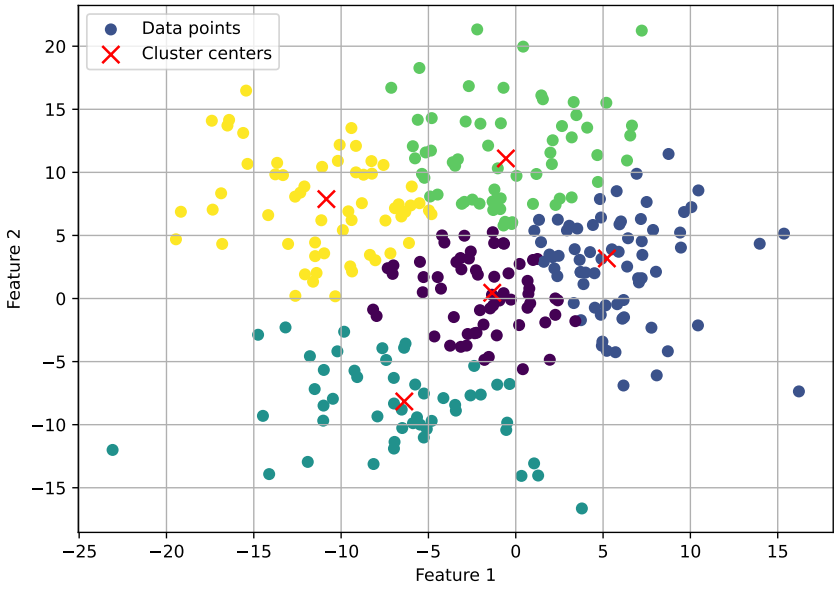

Figure 5: Generated data, cluster assignments and centers with SPSA

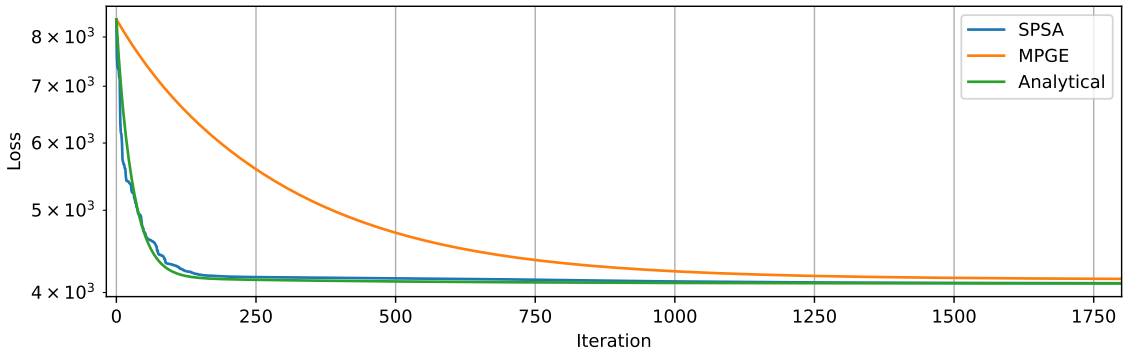

Figure 6: Convergence of SPSA, MPGE and the `Analytical` first-order gradients

A.5 3.2.2

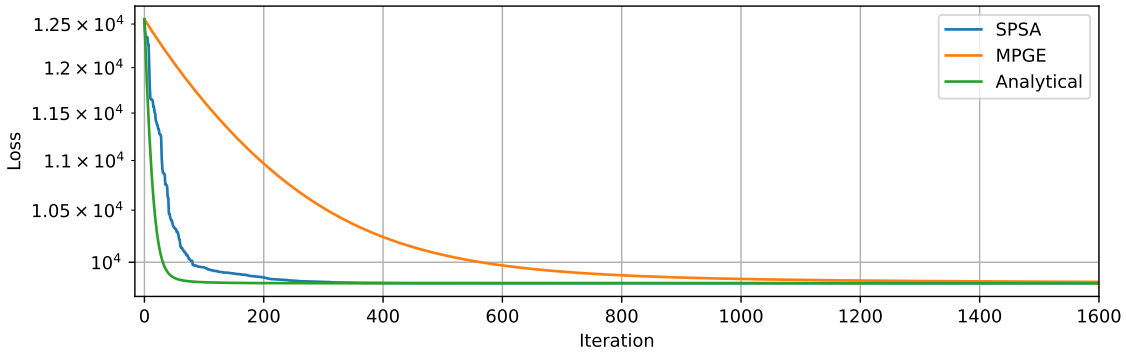

Figure 7: Convergence of SPSA, MPGE and the `Analytical` first-order gradients

A.6 3.3

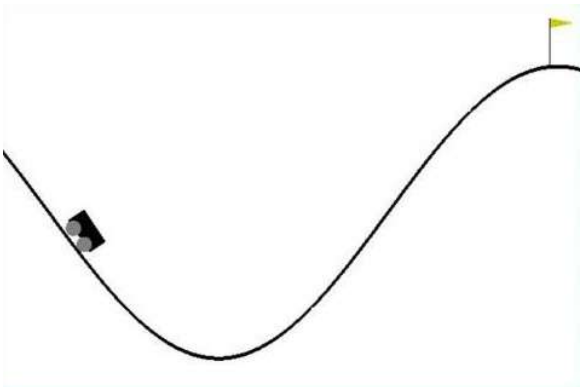

Figure 8: Mountain Car environment, in which the car must build momentum to be able to reach the goal (flag)

A.7 3.3.2

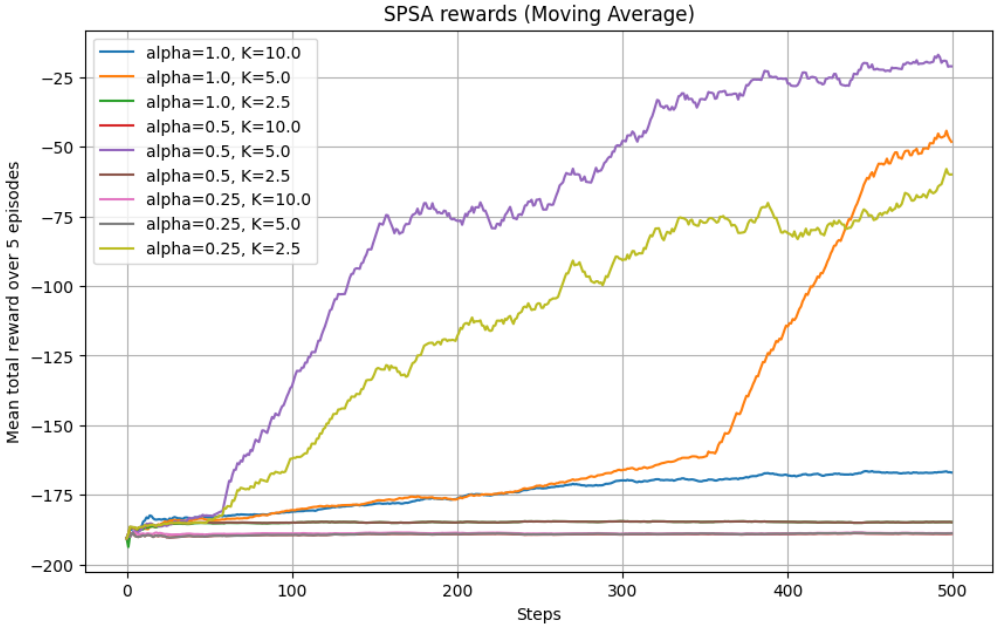

Figure 9: Grid-search training of SPSA in the Mountain Car environment with different combinations of α and K. Higher values means that the algorithm has converged to a good solution, while low values mean that the car never reaches the goal.

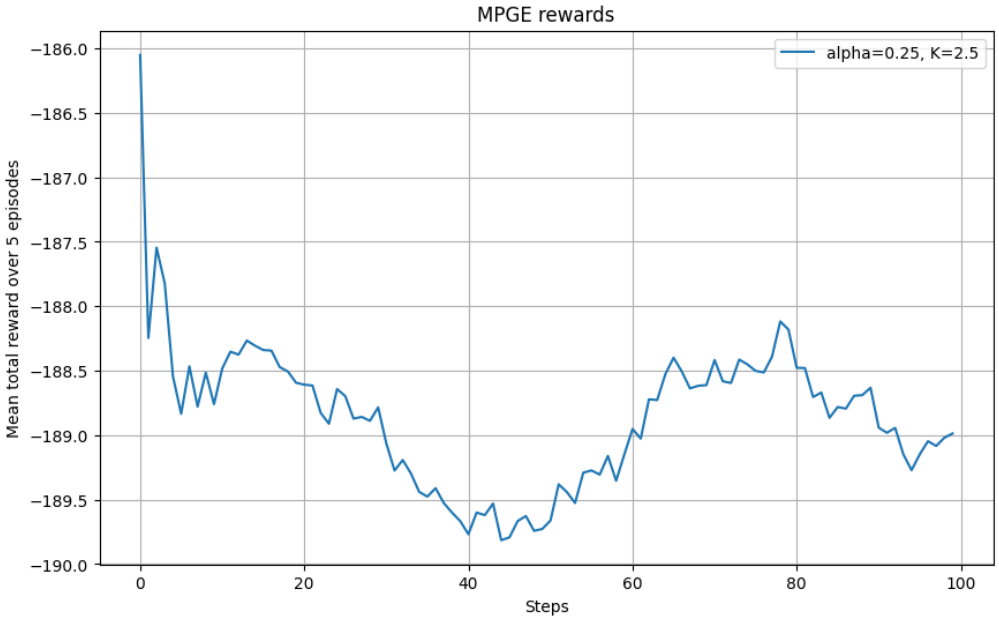

Figure 10: Typical training of MPGE in the mountain car environment. The car never reaches the goal even after a long training.

A.8 3.4.2

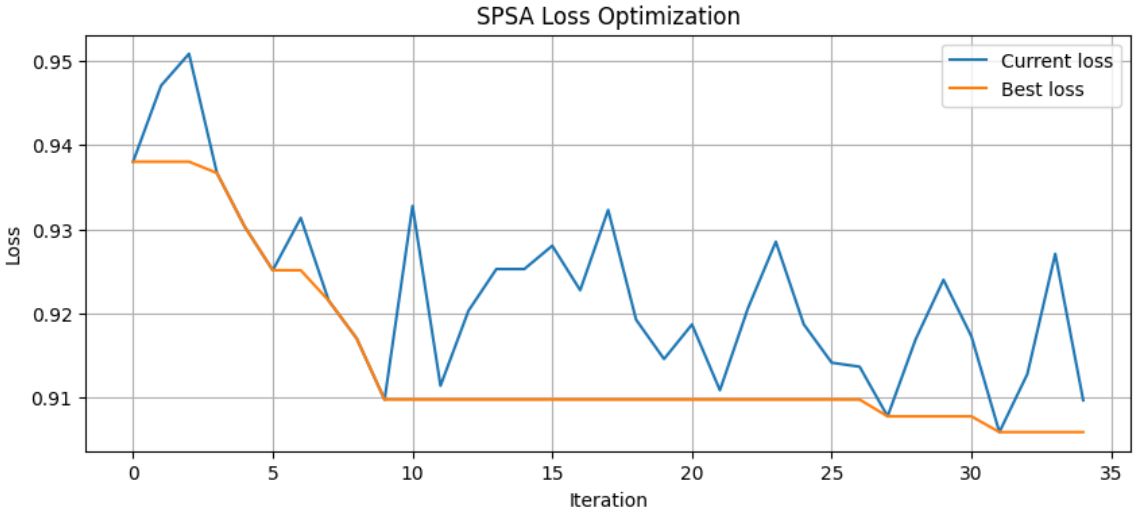

Figure 11: Optimization of the validation loss across iterations, starting from an architecture with a small number of units (131, 131, 51).

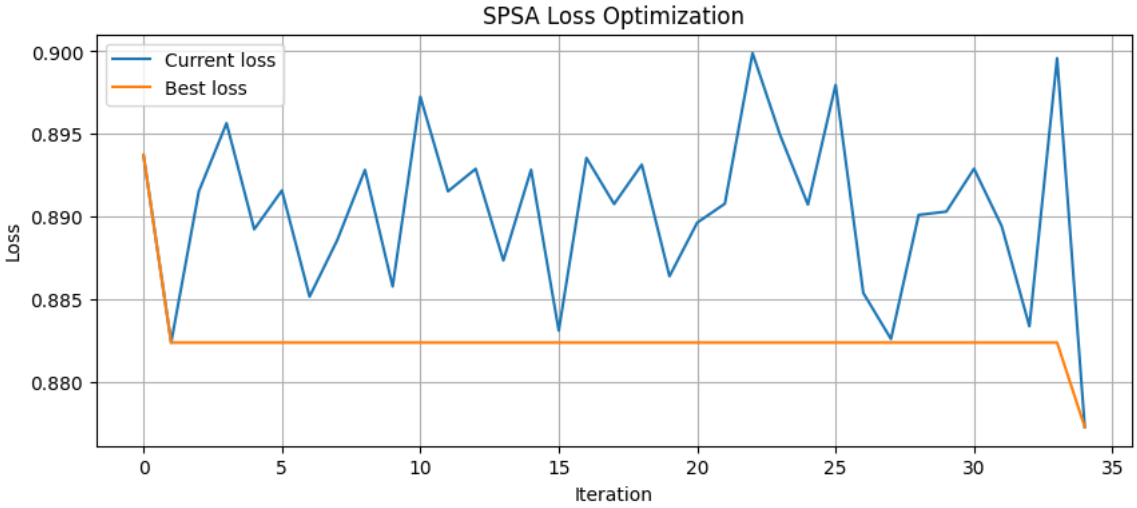

Figure 12: Optimization of the validation loss across iterations, starting from an architecture with a large number of units (528, 528, 256).

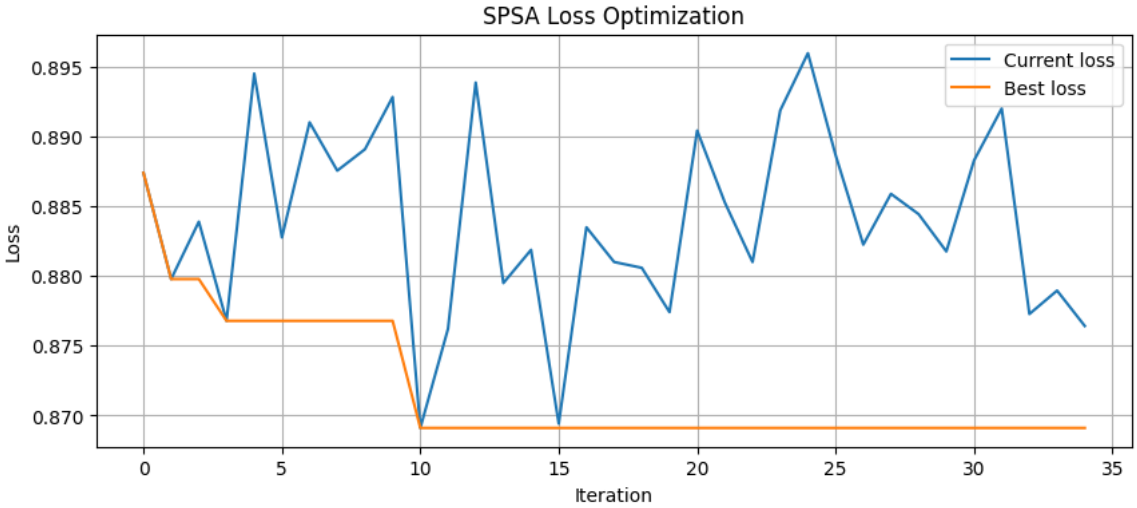

Figure 13: Optimization of the validation loss across iterations, starting from an architecture with a very large number of units (825, 825, 410)

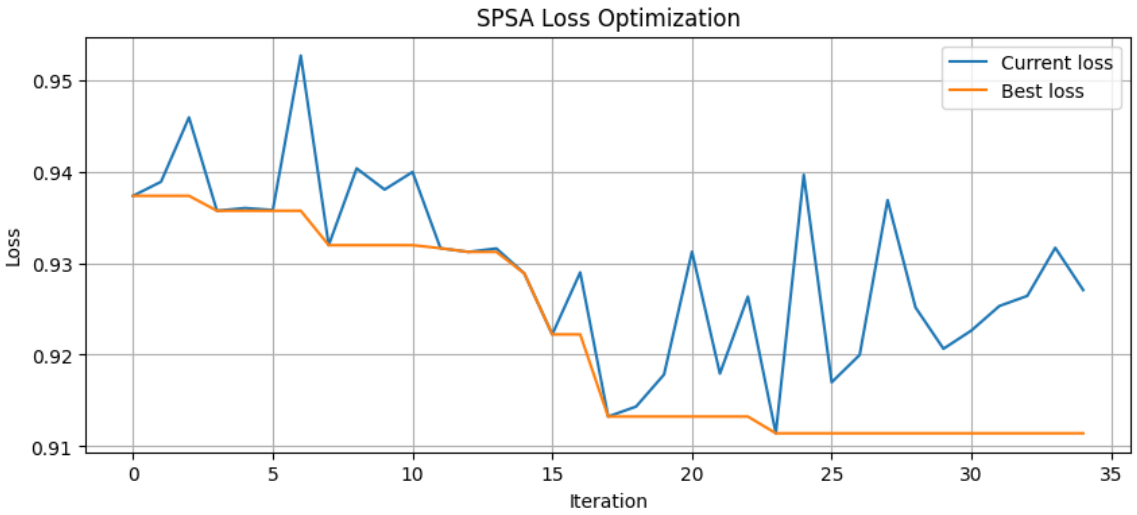

Figure 14: Optimization of the validation loss across iterations, starting from an architecture with a large number of units (528, 528, 256), starting with a smaller learning rate step size.