Improving Evolution Strategies by "learning" over the run

Clement Micol

December 19, 2021

1 Introduction

In this report, we take a look at how the Vanilla ES can be improved by using the information we get of the function over the run. Indeed, at each iteration, we retrieve a good estimation of the gradient of the function and we don't make used of it to compute the next estimation. This information will be used to adapt the sampling of the perturbation which is kept static in the Vanilla ES and lead to poor performance in high dimension. This bad performance in high dimension is due to the high variance of the estimator requiring a lot of sampling to be robust.

The result of this report is adapted from the two articles [MMTS18] and [LLQ20] who present some methods to learn over the run. In their article [MMTS18], Niru Maheswaranathan *et al.* describe how having access to an oracle that return the gradient of the function at a given point can be used to adapt the covariance matrix of the sampling to reduce the variance of the estimation. This article is mainly useful to understand how the standard procedure to adapt the covariance matrix. It's almost the same as the one used for the CMA-ES [Han16] where the best fitted points acts as the surrogate gradients an oracle would return.

2 Vanilla ES :

In this section, we recall the standard algorithm of the Vanilla-ES.

This algorithm is based on the fact that a black-box function f can be made derivable by perturbing it with a Gaussian noise ε. Let $f_\sigma(x) := \mathbb{E}\left[f(x+\varepsilon)\right]$, then we can prove that f is differentiable and that :

$$\nabla f_\sigma(x) = \frac{1}{\sigma}\mathbb{E}\left[f(x+\varepsilon)\varepsilon\right]$$

A more robust estimator of ∇f_σ is the antithetic gradient estimator :

$$\nabla f_\sigma(x) = \frac{1}{2\sigma}\mathbb{E}\left[(f(x+\varepsilon) - f(x-\varepsilon))\varepsilon\right]$$

Using this estimator, we perform a gradient descent to get the optimal value of our function f :

$$x \longleftarrow x - \alpha\nabla f_\sigma(x)$$

Let's get an approximation of the variance of $\hat{g} = \frac{(f(x+\varepsilon)-f(x-\varepsilon))\varepsilon}{2\sigma}$ and to do so we make the approximation :

$$\hat{g} \approx \varepsilon^T \nabla f \varepsilon$$

Then :

$$||\mathbb{E}\left[\hat{g}\right]||^2 = ||\nabla f||^2$$

And :

$$\begin{aligned}
\mathbb{E}[||\hat{g}||^2] &\approx \mathbb{E}[(\nabla f^T \varepsilon)^2 ||\varepsilon||^2] \\
&\approx \sum_i \sum_j \sum_k \mathbb{E}[\varepsilon_k^2 \varepsilon_i \varepsilon_j] \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} \\
&\approx \sum_{i \neq k} \sum_{j \neq k} \sum_k \mathbb{E}[\varepsilon_i \varepsilon_j] \frac{\partial f}{\partial x_i} \frac{\partial f}{\partial x_j} + \sum_k \mathbb{E}[e_k^4] \left(\frac{\partial f}{\partial x_k} \right)^2 \\
&\approx (n-1)||\nabla f||^2 + 3||\nabla f||^2 \\
&\approx (n+2)||\nabla f||^2
\end{aligned}$$

Therefore :

$$\begin{aligned}
\mathrm{Var}[\hat{g}] &= \mathbb{E}[||\hat{g}||^2] - ||\mathbb{E}[\hat{g}]||^2 \\
&\approx (n+2)||\nabla f||^2 - ||\nabla f||^2 \\
&\approx (n+1)||\nabla f||^2
\end{aligned}$$

We can actually prove [CPPH+19] under specific assumption that :

$$|\mathrm{Var}[\hat{g}] - (n+1)||\nabla f||^2| \leq \eta$$

with η as small as we want. Therefore as expected, the variance of the estimator of the Vanilla ES gets very high when the dimension gets high.

Algorithm 1 Vanilla ES

Require: Initial parameter x_0, objective function f, learning rate α, hyper-parameters σ, P and total number of iteration T

for t = 0:T-1 **do**

Sample ε_1,...,ε_P i.i.d. from $\frac{1}{\sqrt{n}}\mathcal{N}(0, I_n)$

$\hat{g} \leftarrow \frac{1}{2\sigma} \sum_{i=1}^{P} (f(x + \varepsilon_i) - f(x - \varepsilon_i))\varepsilon_i$

$x \leftarrow x - \alpha \hat{g}$

end for

3 Guided Gaussian ES :

As we previously suggested, to reduce the variance of the estimator, we will adapt the covariance matrix after each step using our new knowledge on the function. My first idea was to get as close as possible from the algorithm developed in [MMTS18] by updating an oracle that would give at each step a surrogate gradient based on the previous gradient. This oracle is nothing more than a Gaussian regression on the $k << n$ previous states of the gradient. That is to say, we suppose that :

$$\begin{pmatrix} \frac{\partial f^{(t)}}{\partial x_i} \\ \vdots \\ \frac{\partial f^{(t+k+1)}}{\partial x_i} \end{pmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} K_{\alpha,\sigma}(\mathbf{x}, \mathbf{x}) & K_{\alpha,\sigma}(\mathbf{x}, x_{t+k+1}) \\ K_{\alpha,\sigma}(\mathbf{x}, x_{t+k+1}) & K_{\alpha,\sigma}(x_{t+k+1}, x_{t+k+1}) \end{bmatrix}\right)$$

where $K_{\alpha,\sigma}(x,y)$ is the Gaussian kernel :

$$K_{\alpha,\sigma}(x,y) := \alpha e^{-\frac{||x-y||^2}{2\sigma}}$$

α and σ are some hyper-parameters that can be optimized to improve the prediction. But in practice, you want to set those hyper-parameters to some constant to avoid having to recalculate the entire kernel matrix at each step which can be computational costly. You could optimize the hyper-parameters of the kernel for the new updated rows and columns $K_{\alpha,\sigma}(\mathbf{x}, x_{t+k+1})$, $K_{\alpha,\sigma}(\mathbf{x}, x_{t+k+1})$, $K_{\alpha,\sigma}(x_{t+k+1}, x_{t+k+1})$. We can thereby make the following prediction and that's what our oracle will return :

$$\nabla \tilde{f}(t+k+1) := K_{\alpha,\sigma}(\mathbf{x}, x_t+k) K_{\alpha,\sigma}(\mathbf{x},\mathbf{x})^{-1} G_{t+k-1}$$

where G_{t+k-1} is the matrix of historical gradient you kept in memory. Then you would compute your new predicted gradient space :

$$U = \begin{bmatrix} G_{t+k-1} & \nabla \tilde{f}(t+k) \end{bmatrix}$$

and we update the co-variance matrix :

$$C_{t+k+1} = \alpha I_n + (1-\alpha) U U^T$$

The kernel is updated by moving the rows up one notch, moving the columns to the left one notch and calculate the new values $K_{\alpha,\sigma}(\mathbf{x}, x_{t+k+1})$, $K_{\alpha,\sigma}(\mathbf{x}, x_{t+k+1})$, $K_{\alpha,\sigma}(x_{t+k+1}, x_{t+k+1})$. This ensure that maintaining the kernel matrix as a computational cost of $O(k)$.

One key point to prove is that the antithetic gradient estimator is still a descent direction. If $g \sim \mathcal{N}(0, C)$ and the Choleski factorisation of C is $C = LL^T$, than we can compute g by $L\tilde{g}$ where $\tilde{g} \sim \mathcal{N}(0, Id)$.
Let :

$$F_\sigma(\theta) = \mathbb{E}\left[F(\theta + \sigma g)\right] = \mathbb{E}\left[F(\theta + \sigma L \tilde{g})\right]$$

Let's compute $\nabla F_\sigma(\theta)$:

$$\begin{aligned} \frac{\partial F_\sigma}{\partial x_i} =& \lim_{\varepsilon \to \infty} \frac{1}{\varepsilon} \left[F_\sigma(\theta + \varepsilon e_i) - F_\sigma(\theta)\right] && (1) \\ =& \lim_{\varepsilon \to \infty} \frac{1}{(2\pi)^{\frac{1}{n}}} \int_{\mathbb{R}} F(\theta + \sigma L x) e^{-\frac{1}{2}||x||_2^2} \frac{e^{\frac{1}{\sigma}\varepsilon (L^{-1}e_i)^T x - \frac{1}{2}\varepsilon^2 C_{ii}^T} - 1}{\varepsilon} dx && (2) \end{aligned}$$

Since, $\frac{e^{\varepsilon x}-1}{\varepsilon} \leq e^x - 1$ if $\varepsilon \leq 1$ for all $x \in \mathbb{R}$, we can used dominated convergence theorem to justify that :

$$\lim_{\varepsilon \to \infty} \frac{1}{(2\pi)^{\frac{1}{n}}} \int_{\mathbb{R}} F(\theta + Lx) e^{-\frac{1}{2}||x||_2^2} \frac{e^{\frac{1}{\sigma}\varepsilon (L^{-1}e_i)^T x - \frac{1}{2}\varepsilon^2 C_{ii}^T} - 1}{\varepsilon} dx = \frac{1}{(2\pi)^{\frac{1}{n}}} \int_{\mathbb{R}} \frac{1}{\sigma} (L^{-1}e_i)^T x F(\theta + Lx) e^{-\frac{1}{2}||x||_2^2} dx$$

That is to say :

$$\frac{\partial F_\sigma}{\partial x_i} = \frac{1}{(2\pi)^{\frac{1}{n}}} \int_{\mathbb{R}} \frac{1}{\sigma} (L^{-1}e_i)^T x F(\theta + Lx) e^{-\frac{1}{2}||x||_2^2} dx$$

And therefore :

$$\nabla F_\sigma(\theta) = \frac{1}{\sigma} C^{-1} \mathbb{E}\left[F(\theta + \sigma g) g\right]$$

We deduce that :

$$\frac{1}{2\sigma} \mathbb{E}[(f(x + \sigma g) - f(x - \sigma g)) g] = C \nabla F_\sigma(x)$$

Since $C \geq 0$, the antithetic gradient estimator is still a gradient descent. Now let's have a look at the variance of this estimator. As for the last section we'll do the following approximation :

$$\hat{g} = \frac{1}{2\sigma}(f(x+\sigma\varepsilon) - f(x-\sigma\varepsilon))\varepsilon \approx \varepsilon^T \nabla f \varepsilon$$

This approximation enable us to easily calculate :

$$\begin{aligned}
\mathbb{E}[||\hat{g}||^2] &\approx \mathbb{E}[||\varepsilon^T \nabla f \varepsilon||^2] \\
&\approx \sum_k \sum_i \sum_j \mathbb{E}[\varepsilon_k^2 \varepsilon_i \varepsilon_j] g_i g_j \\
&\approx \sum_k \sum_i \sum_j (C_{kk} C_{ij} + 2C_{ki} C_{kj}) g_i g_j \\
&\approx \sum_k C_{kk} g^T C g + 2 g^T C^2 g
\end{aligned}$$

Moreover :

$$||\mathbb{E}[\hat{g}]||^2 = g^T C^2 g$$

Thus :

$$\mathrm{Var}[\hat{g}] \approx \sum_k C_{kk} g^T C g + g^T C^2 g$$

In the case where $C = \alpha I_n + (1-\alpha) G G^T$, $C_{kk} = \alpha + (1-\alpha)\sum_i (g_k^i)^2$ and $C^2 = \alpha^2 I_n + (2-\alpha)\alpha G G^T$. Therfore, we compute that :

$$\mathrm{Var}[\hat{g}] \approx \alpha[\alpha(n+1-k)+k] \times ||g||^2 + [\alpha(n+2-2k) - \alpha^2(n+1-k)+k] \times ||G^T g||^2$$

where I recall k is the number of historic gradients in your gradient space. We see that the smaller the value of α the smaller the variance is to a point where :

$$\lim_{\alpha \to 0} \mathrm{Var}[\hat{g}] = k \times ||G^T g||^2 << n \times ||g||^2$$

This highlights the existence of a trade-off with the α parameter : if α is to small, we might get stuck in a gradient space that doesn't the true gradient space (lack of exploration) but if α is to high, too much points are being explored and we don't exploit our knowledge of the function. This trade-off is the core idea of the next algorithm presented in the next section.

Postscript : Actually the Gaussian regression only works if you kernel is optimized. I was misled by the performance of the QR-decomposition of the gradient space. I think that this method though it works, is to computational expensive.

4 Self-Guided ES:

As explained in the previous section, we can drastically reduce the variance of the estimator by updating over each iteration the coefficient α. As a reminder, α correspond to the weight of the gradient subspace $\mathcal{G}$ in the definition of the co-variance matrix :

$$C = \alpha G G^T + (1-\alpha) I_n$$

Therefore, the Self-Guided ES developed in [LLQ20] tends to control the parameter α such that if the gradient subspace is "good", α is close to 1 (we exploit) and if the current gradient subspace doesn't represent accurately the future gradients, α should be taken close to 0 (we

Algorithm 2 Gaussian Guided ES

Require: Initial parameter x_0, objective function f, number of historic gradients kept k, learning rate η, hyper-parameters σ, P and total number of iteration T

Create kernel $K = 0$

for t = 0:T-1 **do**

if $t < k$ **then**

Update kernel by calculating $K(\mathcal{X}, x_t)$ and $K(x_t, x_t)$ (and optimize the parameters ?)

Sample ε_1,...,ε_P i.i.d. from $\frac{\sigma}{\sqrt{n}}\mathcal{N}(0, I_n)$

$\hat{g} \leftarrow \frac{1}{2\sigma}\sum_{i=1}^{P}(f(x+\varepsilon_i) - f(x-\varepsilon_i))\varepsilon_i$

$x \leftarrow x - \alpha\hat{g}$

Add x and $\hat{g}$ to the historic state space and $\mathcal{X}$ to the historic gradient space $\mathcal{G}$ respectively

else

Update kernel by calculating $K(\mathcal{X}, x_t)$ and $K(x_t, x_t)$ (and optimize the parameters ?)

$\tilde{g}_{t+1} \leftarrow K(\mathcal{X}, x_t)K(\mathcal{X}, \mathcal{X})^{-1}G$

$U \leftarrow \begin{bmatrix} G & \tilde{g}_{t+1} \end{bmatrix}$

Get the QR-decomposition of U

Sample $\varepsilon_1^{(f)}$,...,$\varepsilon_P^{(f)}$ i.i.d. from $\sigma\sqrt{\frac{\alpha}{n}}\mathcal{N}(0, I_n)$

Sample $\varepsilon_1^{(g)}$,...,$\varepsilon_P^{(g)}$ i.i.d. from $\sigma\sqrt{\frac{1-\alpha}{k}}\mathcal{N}(0, I_k)$

For all i, $\varepsilon_i \leftarrow \varepsilon_i^{(f)} + Q\varepsilon_i^{(g)}$

$\hat{g} \leftarrow \frac{1}{2\sigma}\sum_{i=1}^{P}(f(x+\varepsilon_i) - f(x-\varepsilon_i))\varepsilon_i$

$x \leftarrow x - \eta\hat{g}$

Add x and $\hat{g}$ to the historic state space and $\mathcal{X}$ to the historic gradient space $\mathcal{G}$ respectively

end if

end for

explore).
In this algorithm, we keep track of two subspaces : the gradient subspace $\mathcal{G}$ and its orthogonal $\mathcal{G}^{\perp}$. Suppose $G \in \mathcal{M}_{n\times k}$ corresponds to the matrix of historic gradient. Let's get the Singular Value Decomposition of G :

$$\exists U \in \mathcal{O}(n), \exists S \in \text{diag}(m), \exists V \in \mathcal{O}(m) \text{ such that :}$$

$$G = U \begin{bmatrix} S & 0 \end{bmatrix} V^T$$

where the diagonal of S corresponds to the singular value of G. Suppose the singular value are sorted in decreasing order $s_1 > s_2 > ... > s_m$ and let $\tilde{m} = \max\{n \in \{1, ..., m\} | s_n > 0\}$. Then it comes that the gradient subspace is generated by $U_1, U_2, ..., U_{\tilde{m}}$ and its orthogonal by $U_{\tilde{m}+1}, ..., U_n$ where U_i is the i^{th} column of U. This can be sum up by :

$$\mathbb{R}^n = \underbrace{\text{Vect}(U_1, U_2, ..., U_{\tilde{m}})}_{=\mathcal{G}} + \underbrace{\text{Vect}(U_{\tilde{m}+1}, ..., U_n)}_{=\mathcal{G}^{\perp}}$$

We then sample the search direction ε_1,...,ε_P using a hybrid distribution on these two subspaces :

$$\begin{cases} \varepsilon_i \sim \mathcal{N}(0, I_{\mathcal{G}}) \text{ with probability } \alpha \\ \varepsilon_i \sim \mathcal{N}(0, I_{\mathcal{G}^{\perp}}) \text{ with probability } 1 - \alpha \end{cases}$$

Let's elaborate a bit more on how the research directions are obtained, and especially what sampling on $\mathcal{N}(0, I_{\mathcal{G}})$ means. First of all, we should notice that for sampling computationally faster, one should not sample a random variables B_i drawn from a Bernoulli of success probability α at each steps and decide if ε_i is sample from $\mathcal{N}(0, I_{\mathcal{G}})$ or $\mathcal{N}(0, I_{\mathcal{G}^{\perp}})$. Instead, we should generate a random variables N drawn from a binomial distribution of probability of success α and a population size of P. Then N corresponds to the number of research directions we should sample from $\mathcal{N}(0, I_{\mathcal{G}})$ and $n-N$ is the number of research directions we should sample from $\mathcal{N}(0, I_{\mathcal{G}^{\perp}})$.
Now let's retake our previous notation and let :

$$\mathcal{G} = \text{Vect}(U_1, ..., U_{\tilde{m}})$$

$$\mathcal{G}^{\perp} = \text{Vect}(U_{\tilde{m}+1}, ..., U_n)$$

Then, if $U_{\mathcal{G}} = \begin{bmatrix} U_1 & \cdots & U_{\tilde{m}} \end{bmatrix}$, then $I_{\mathcal{G}} = U_{\mathcal{G}} U_{\mathcal{G}}^T$. Since, $U \in \mathcal{O}(n)$, $I_{\mathcal{G}}$ has the following spectral decomposition :

$$U_{\mathcal{G}} U_{\mathcal{G}}^T = U \begin{bmatrix} I_{\tilde{m}} & 0 \\ 0 & 0 \end{bmatrix} U^T$$

Therefore, $\varepsilon_i \sim \mathcal{N}(0, I_{\mathcal{G}})$ can be sampled by :

$$\varepsilon_i \sim U \begin{bmatrix} I_{\tilde{m}} & 0 \\ 0 & 0 \end{bmatrix} \mathcal{N}(0, I_n)$$

That is to say :

$$\varepsilon_i \sim U_{\mathcal{G}} \mathcal{N}(0, I_k)$$

Similarly, if ε_i should be drawn from the orthogonal subspace $\mathcal{G}^{\perp}$, then :

$$\varepsilon_i \sim U_{\mathcal{G}^{\perp}} \mathcal{N}(0, I_{n-k})$$

Once we have sampled our research directions, we can update the hybrid distribution $\mathcal{B}(P, \alpha)$. The best way to assess which of the two subspaces was most fitting is to look at the ratio

$\alpha_t = \frac{||\nabla f_\sigma^{\mathcal{G}}||^2}{||\nabla f_\sigma^{\mathcal{G}^\perp}||^2}$. If α_t is greater than 1 that would mean that we should research the next gradient in the orthogonal subspace, while on the other hand if α_t is smaller than 1 that would mean that the next gradient is generated by the current gradient subspace. As in [LLQ20], we introduce the quantity $\hat{r}_G$ and $\hat{r}_{G^\perp}$ to regulate the parameter α where :

$$\hat{r}_G = \frac{1}{N}\sum_{i=1}^{N} \min(f(x+\varepsilon_i), f(x-\varepsilon_i))$$

$$\hat{r}_G^\perp = \frac{1}{P-N}\sum_{i=N+1}^{P} \min(f(x+\varepsilon_i), f(x-\varepsilon_i))$$

where N is the binomial random variable that decides the number of research directions that are being sampled from the subspace $\mathcal{G}$. It naturally comes that α is updated by the following set of equations :

$$\alpha = \begin{cases} \min(\delta\alpha, k_1) \text{ if } \hat{r}_G < \hat{r}_{\mathcal{G}^\perp} \text{ or } \hat{r}_{\mathcal{G}^\perp} \text{ doesn't exist} \\ \max(1/\delta\alpha, k_2) \text{ if } \hat{r}_{\mathcal{G}^\perp} < \hat{r}_G \text{ or } \hat{r}_{\mathcal{G}} \text{ doesn't exist} \end{cases}$$

where $\delta > 1$ is a scaling factor, and k_1,k_2 are the upper and lower bound of α.

Let's have a look at how this hybrid sampling distribution reduce the variance of the antithetic estimator. First, we should notice that :

$$C = \mathbb{E}_{\epsilon\sim\mathcal{B}(\alpha)}[\epsilon\epsilon^T] = \alpha U_{\mathcal{G}} U_{\mathcal{G}}^T + (1-\alpha) U_{\mathcal{G}^\perp} U_{\mathcal{G}^\perp}^T$$

Then we recall from last section that under the approximation $\hat{g} \approx \varepsilon^T g \varepsilon$, we have :

$$\text{Var}[\hat{g}] \approx \sum_j C_{kk} g^T C g + g^T C^2 g$$

But :

$$C_{kk} = \alpha \sum_{i=1}^{\tilde{m}} (u_{i,\mathcal{G}}^k)^2 + (1-\alpha) \sum_{i=1}^{n-\tilde{m}} (u_{i,\mathcal{G}^\perp}^k)^2$$

And therefore :

$$\sum_k^n C_{kk} = \alpha \sum_{i=1}^{\tilde{m}} ||u_{i,\mathcal{G}}||^2 + (1-\alpha) \sum_{i=1}^{n-\tilde{m}} ||u_{i,\mathcal{G}^\perp}||^2 = \tilde{m}\alpha + (1-\alpha)(n-\tilde{m})$$

Moreover, since $U_{\mathcal{G}} U_{\mathcal{G}}^T U_{\mathcal{G}^\perp} U_{\mathcal{G}^\perp}^T = 0$, we have :

$$C^2 = \alpha^2 U_{\mathcal{G}} U_{\mathcal{G}}^T + (1-\alpha)^2 U_{\mathcal{G}^\perp} U_{\mathcal{G}^\perp}^T$$

Thus, we can approximate the variance of antithetic estimator by :

$$\text{Var}[\hat{g}] = [(\tilde{m}+1)\alpha + (1-\alpha)(n-\tilde{m})]\alpha ||U_{\mathcal{G}}^T g||^2 + [\tilde{m}\alpha + (1-\alpha)(n+1-\tilde{m})](1-\alpha)||U_{\mathcal{G}^\perp}^T g||^2$$

We can then find for which parameter α_e, $\text{Var}[\hat{g}]$ reaches one of its extremum :

$$\alpha_e = \frac{(m-n)||U_{\mathcal{G}}^T g||^2 + [2(n+1)-3m]||U_{\mathcal{G}^\perp}||^2}{(2m+1-n)||U_{\mathcal{G}}^T g||^2 + 2(n+1-2m)||U_{\mathcal{G}^\perp}^T g||^2}$$

We can not say in advance if it is a maximum or a minimum it depends on $||U_{\mathcal{G}}^T g||^2$ and $||U_{\mathcal{G}^\perp}^T g||^2$). Indeed :

$$\frac{\partial^2}{\partial \alpha^2}\text{Var}[\hat{g}] = 2\underbrace{(2m+1-n)}_{<0}||U_{\mathcal{G}}^T g||^2 + \underbrace{2(n+1-2m)}_{>0}||U_{\mathcal{G}^\perp}^T g||^2 \tag{3}$$

$$\sim n(||U_{\mathcal{G}^\perp}^T g||^2 - ||U_{\mathcal{G}}^T g||^2) \tag{4}$$

Thus $\frac{\partial^2}{\partial \alpha^2}\text{Var}[\hat{g}] > 0$ if $||U_{\mathcal{G}^\perp}^T g||^2 > ||U_{\mathcal{G}}^T g||^2$ which mean that α_e corresponds to the minimum of variation. It is logical that in this case, α_e would corresponds to the minimum of variation since we need to exploit instead of exploring. On the other hand, if $||U_{\mathcal{G}^\perp}^T g||^2 < ||U_{\mathcal{G}}^T g||^2$, α_e correspond to the maximum of variation. Indeed, for this case we need to explore more than exploiting since the orthogonal subspace is better than the gradient subspace.

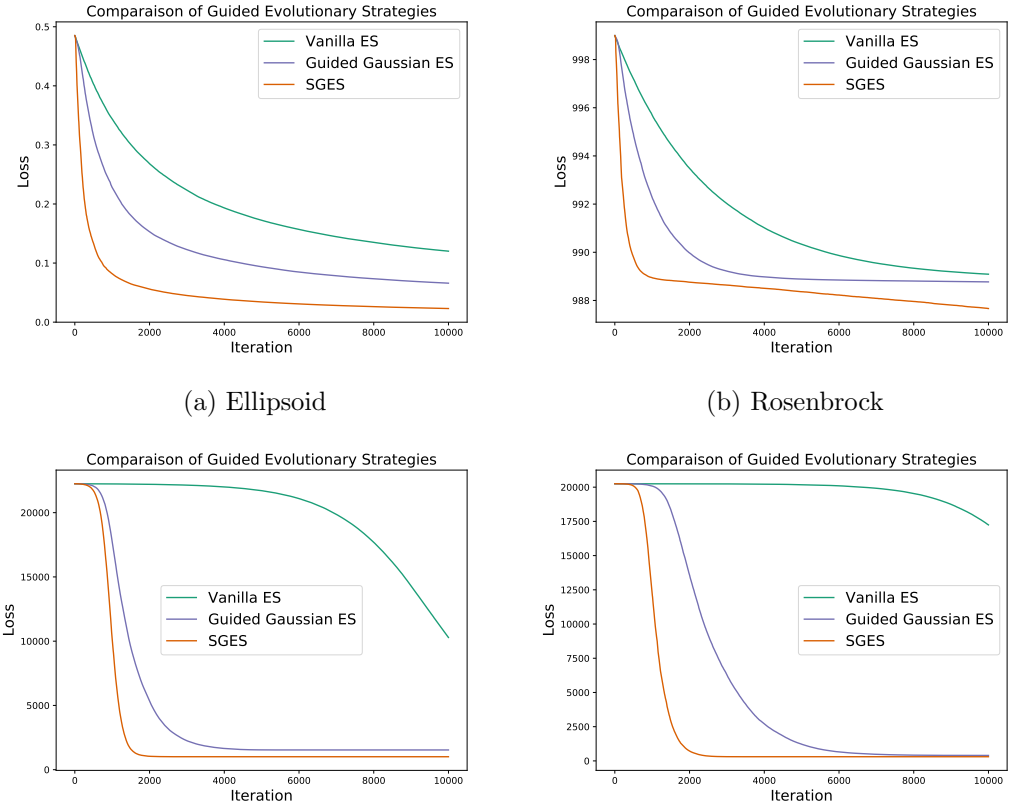
We can see that overall, when α is close to 1, $\text{Var}[\hat{g}] \approx (\tilde{m}+1)||U_{\mathcal{G}}^T g||^2$ and since $\tilde{m} << n$, $\text{Var}[\hat{g}] << (n+1)||g||^2$. This process reduces indeed the variance of the antithetic estimator in comparison to the Vanilla ES.

Algorithm 3 Self-Guided ES

Require: Initial parameter x_0, objective function f, learning rate η, hyper-parameters σ, P, k, δ, α_0 and total number of iteration T
Initialize the historical gradient matrix G
for t = 0:T-1 **do**
  if t¡k **then**
    Sample ε_1,...,ε_P i.i.d. from $\frac{1}{\sqrt{n}}\mathcal{N}(0, I_n)$
    $\hat{g} \leftarrow \frac{1}{2\sigma}\sum_{i=1}^{P}(f(x+\varepsilon_i) - f(x-\varepsilon_i))\varepsilon_i$
    $x \leftarrow x - \eta\hat{g}$
    Assign $\hat{g}$ to the t^{th} column of G
  else
    Get the SVD decomposition U, S, V of G
    Get $U_{\mathcal{G}}$ and $U_{\mathcal{G}^\perp}$ from the singular values and U
    Sample N from a binomial law of parameter (P, α)
    Sample $\varepsilon_1, ..., \varepsilon_N$ from $\frac{1}{\sqrt{k}}U_{\mathcal{G}}\mathcal{N}(0, I_k)$
    Sample $\varepsilon_{N+1}, ..., \varepsilon_P$ from $\frac{1}{\sqrt{n-k}}U_{\mathcal{G}^\perp}\mathcal{N}(0, I_{n-k})$
    $\hat{g} \leftarrow \frac{1}{2\sigma}\sum_{i=1}^{P}(f(x+\varepsilon_i) - f(x-\varepsilon_i))\varepsilon_i$
    $x \leftarrow x - \eta\hat{g}$
    Add $\hat{g}$ to the historical gradient matrix G
    Adapt α by computing $\hat{r}_G$ and $\hat{r}_{G^\perp}$
  end if
end for

5 Experiments :

To evaluate the performance of the Guided Gaussian ES and the Self-Guided ES, we performed experiments on the Nevergrad functions as well as on some continuous MuJoCo locomotion tasks from the OpenAI Gym library.



(a) Ellipsoid

(b) Rosenbrock

(c) Rastrigin

(d) Lunacerk

Figure 1: Comparison of the guided evolution strategies

5.1 Nevergrad Black-box Functions :

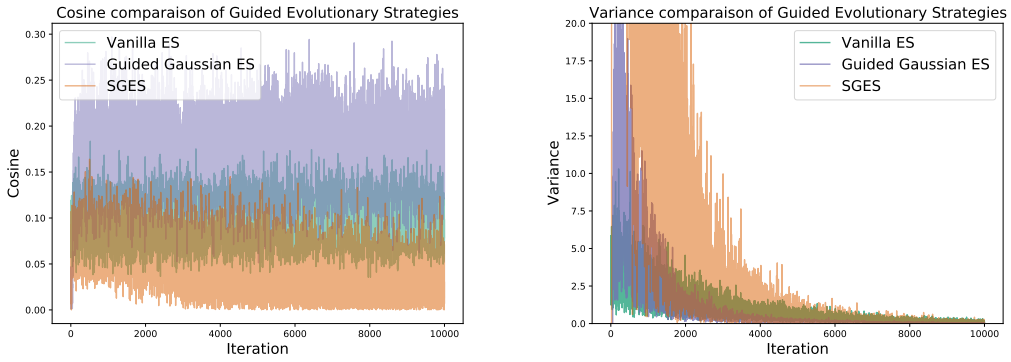
Let's compare first the performance of the Vanilla ES, the Guided Gaussian ES and the Self-Guided ES on the ellipsoid function, the Rosenbrock function, the Rastrigin function and the Lunacerk function. Figure 1 was obtained by minimizing those Nevergrad functions in dimension 1000. The common hyper-parameters are the same for each ES. For the first case, the ellipsoid was optimized with $P = 10$, $\sigma = 0.1$, $k = 20$ and a learning rate of 0.1. For the Rosenbrock function, we used a learning rate of 0.01, $P = 10$, $\sigma = 0.1$ and $k = 20$. For the Lunacerk and the Rastrigin function we used both a higher σ of 1 and a smaller learning-rate of 0.001. This is due to the abundance of local extrema. Also we increased the population size P to 50 for the Rastrigin function. A population size of $P = 10$ should work as well. If the learning-rate is not taken small enough for those two functions, either the ES doesn't converge or it converge toward a local minimum. For theses cases σ can be set to a small value

function	Vanilla ES	Guided Gaussian ES	Self-Guided ES
Ellipsoid	1min 11s	2min 12s	3min 57s
Rosenbrock	45.3s	1min 51s	3min 10s
Rastrigin	58.2s	2min 5s	3min 34s
Lunacerk	1min 8s	2min 9s	3min 53s

Table 1: Running time of each ES for different functions

since there is no noise. But in the case where the function are noisy as we will see for the RL

tasks, we need σ to be higher (of the order of 1 or 2). Figure 2 shows an unexpected result



(a) (b)

Figure 2: (a) Cosine similarity between the estimated gradient and the true gradient and (b) is the variance of the estimated gradient for the ellipsoid function.

since we were expecting as in [LLQ20] the variance of the SGES to be below the variance of the ES Vanilla (except in [LLQ20] they are considering the sphere function and not the ellipsoid). Instead, the variance starts very but converges extremely fast toward 0. This may be a benefit since at first we don't know where to go and we explore and lot and once we have found the right path we stick with it. However, it's difficult to explain why the cosine similarity for the SGES is so low for this function. This difference of result may be cause by the re-normalization by $\sqrt{1/n + 1/k}$.

5.2 Reinforcement Learning Tasks :

As [SHCS17] mention in their article, Evolution Strategies can be a good alternative to Reinforcement Learning algorithm. Typically a reinforcement learning task as the following structure. You define first an environment where your workers (or agents) will interact with. Each worker has define state $\mathcal{S}$ in this environment and he is offer a set of possible actions $\mathcal{A}$ he can take to maximize his instantaneous reward. The goal of the Reinforcement Learning algorithm is to find a Policy network taking as input the state of a worker and return the action he should make to maximize our reward. If we can read between the line, we can see that this problem can be formulated as the following optimization. Let $\mathcal{P}$ be our policy variables (the variables of our MLP) and $f(P)$ our total reward over a long-run with a random initial state $s_0 \in \mathcal{S}$. Therefore with this notation, a reinforcement task can be formulated as :

$$\text{Find } P^* \text{ such that :}$$

$$P^* = \underset{P \in \mathbb{R}^n}{\operatorname{argmax}} f(P)$$

f is a black-box function who return the total reward under policy P with a noise. Indeed since the initial state is random, for the same policy P, $f(P)$ can return two different values. Therefore, the algorithm we developed above seems like the perfect answer to this Problem. In my case, I only add the time to find the optimal solution for the Cart-Pole task of open Gym AI. But I guess, that for other tasks the thought is the same expect you have to modify your MLP (adding edges and node to the network for instance). For the Cart-Pole, I ended up with a MLP with to layer where the first one as 4 inputs (the dimension of the state space $\mathcal{S}$) and 2 outputs (the dimension of the action space $\mathcal{A}$). The activation function for this layer is

a Rectified Linear Unit function. For the second layer, I have two outputs and the activation is the softmax function to enable our network to classified our set of actions. Therefore, the total dimension of our problem is 26. Figure 3 was obtained by taking a learning rate of 0.05, a σ of 1 and a population size P of 20 over 5 different seeds.

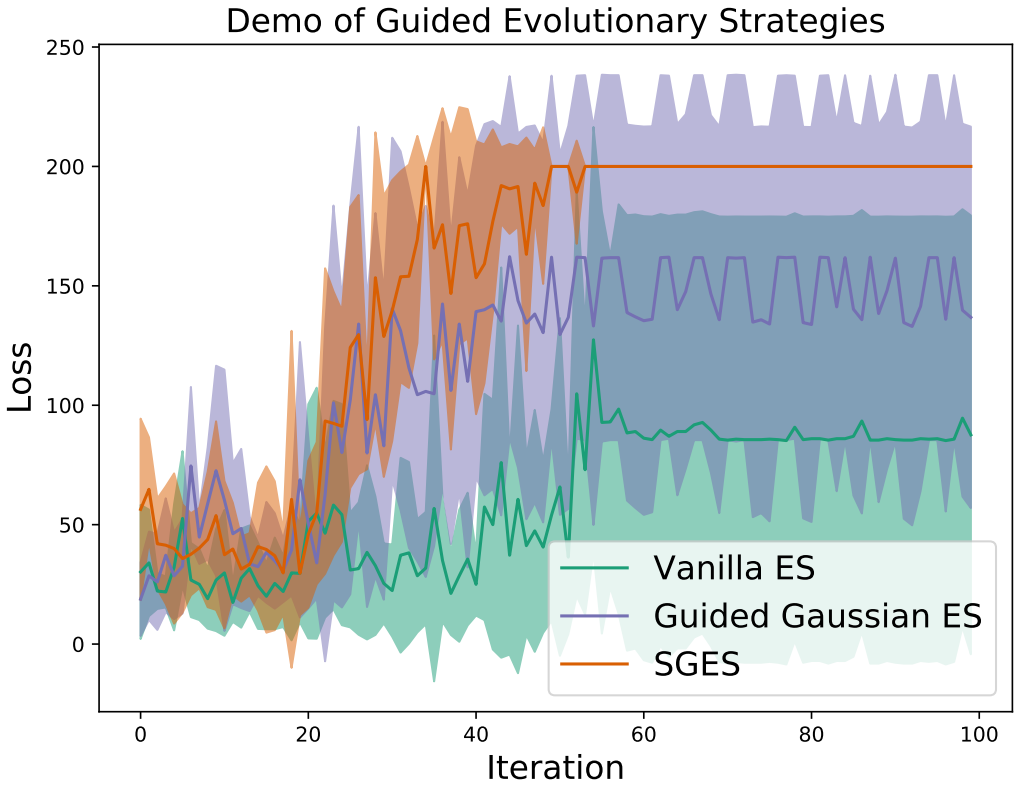


Figure 3: Comparison of the ES Algorithm on the CartPoleV0 task

6 Conclusion :

We saw that the SGES of [LLQ20] and the used of the historic gradients for the Guide Gaussian ES enable us to tackle the problem of high variance present in the Vanilla ES. Search directions are sampled from either the gradient subspace built from the Singular Values Decomposition of the Historic gradient matrix or its orthogonal according to a binomial distribution of parameter α and P. The probability of success of this distribution is updated according to the result of the exploration : if the gradient space is good, we encourage sampling from it, otherwise we encourage sampling from its orthogonal. Empirical results on the Nevergrad black-box functions as well as Cart Pole Reinforcement learning tasks show the excellent performance of SGES and of Guided Gaussian Regression. Due to a lack of time, I couldn't experiment on the Mujoco task and I would interested to see how my algorithm perform on this problem.

References

[CPPH+19] Krzysztof M Choromanski, Aldo Pacchiano, Jack Parker-Holder, Yunhao Tang, and Vikas Sindhwani. From complexity to simplicity: Adaptive es-active subspaces for blackbox optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[Han16] Nikolaus Hansen. The CMA evolution strategy: A tutorial. *CoRR*, abs/1604.00772, 2016.

[LLQ20] Fei-Yu Liu, Zi-Niu Li, and Chao Qian. Self-guided evolution strategies with historical estimated gradients. In Christian Bessiere, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 1474–1480. International Joint Conferences on Artificial Intelligence Organization, 7 2020. Main track.

[MMTS18] Niru Maheswaranathan, Luke Metz, George Tucker, and Jascha Sohl-Dickstein. Guided evolutionary strategies: escaping the curse of dimensionality in random search. *CoRR*, abs/1806.10230, 2018.

[SHCS17] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. 03 2017.