Low-Variance Gradient Estimation in Unrolled Computation Graphs with ES-Single

Paul Vicol [1]

Abstract

We propose an evolution strategies-based algorithm for estimating gradients in unrolled computation graphs, called ES-Single. Similarly to the recently-proposed Persistent Evolution Strategies (PES), ES-Single is unbiased, and overcomes chaos arising from recursive function applications by smoothing the meta-loss landscape. ES-Single samples a single perturbation per particle, that is kept fixed over the course of an inner problem (e.g., perturbations are not re-sampled for each partial unroll). Compared to PES, ES-Single is simpler to implement and has lower variance: the variance of ES-Single is constant with respect to the number of truncated unrolls, removing a key barrier in applying ES to long inner problems using short truncations. We show that ES-Single is unbiased for quadratic inner problems, and demonstrate empirically that its variance can be substantially lower than that of PES. ES-Single consistently outperforms PES on a variety of tasks, including a synthetic benchmark task, hyperparameter optimization, training recurrent neural networks, and training learned optimizers.

1. Introduction

Many problems in machine learning involve computing gradients through unrolled computation graphs, including bilevel optimization (such as hyperparameter optimization (Domke, 2012; Maclaurin et al., 2015; Franceschi et al., 2017; Shaban et al., 2019) and meta-learning (Bertinetto et al., 2018; Finn, 2018; Finn et al., 2018)), RNN training (Merity et al., 2018), reinforcement learning (Salimans et al., 2017; Mania et al., 2018), and training learned optimizers (Metz et al., 2019; 2018; 2020a;b; Wichrowska et al., 2017; Andrychowicz et al., 2016; Li & Malik, 2016; 2017). In each of these tasks, we wish to learn parameters that govern the evolution of a dynamical system, such that the states produced by the system satisfy some objective function. For example, in hyperparameter optimization, we aim to tune hyperparameters (e.g., the learning rate or dropout coefficient), such that a model trained using these hyperparameters achieves low loss—in this case, the hyperparameters govern the evolution of the model parameters, that can be interpreted as the states of the dynamical system. Classic approaches to computing gradients through unrolled computation graphs include reverse-mode (Werbos, 1990) and forward-mode (Williams & Peng, 1990; Franceschi et al., 2017; Tallec & Ollivier, 2017a; Menick et al., 2021) gradient accumulation.

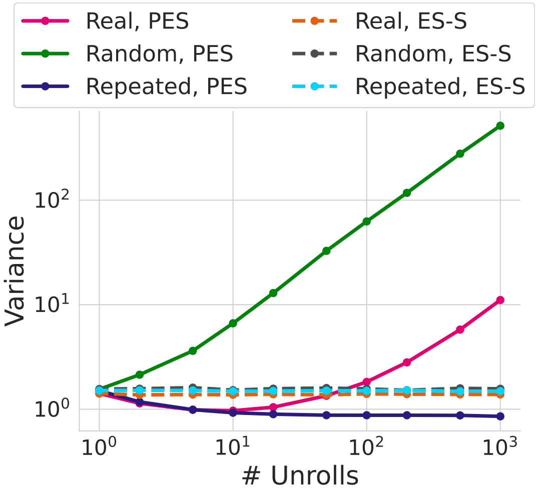


Figure 1. Comparing empirical variance measurements for PES (solid curves) and ES-Single (ES-S, dashed lines) on a small LSTM training task. Unlike PES, under all conditions the variance of ES-Single is constant as the number of partial unrolls increases.

However, gradient-based methods face a fundamental obstacle: the loss landscape resulting from long unrolls is often chaotic and can exhibit near discontinuities (Metz et al., 2019; Parmas et al., 2018; Parmas & Sugiyama, 2019), rendering gradients unsuitable (Metz et al., 2021). One approach to overcome this chaos is to consider Gaussian smoothing of the outer loss surface; the gradient of such a smoothed objective can be computed using evolution strategies (ES) (Rechenberg, 1973). Applying vanilla ES to the

[1] Google Brain. Correspondence to: Paul Vicol <paulvicol@google.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023. Copyright 2023 by the author(s).

full unrolled inner problem yields a useful gradient estimate, but leads to slow outer optimization; in contrast, applying ES to truncated unrolls leads to truncation bias, similarly to truncated backpropagation through time.

Vicol et al. (2021) proposed an ES-based algorithm called Persistent Evolution Strategies (PES), that yields unbiased gradient estimates from truncated unrolls, speeding up meta-optimization by allowing for more frequent outer parameter updates. PES has a number of desirable characteristics, including unbiasedness and Gaussian smoothing of the outer loss landscape. However, its variance increases with the number of truncated unrolls per full inner problem, potentially making it impractical to use short truncations for long-horizon problems (for example, truncations of length $K = 1$ for problems where $T \geq 100$).

In this paper, we propose an unbiased gradient estimator for unrolled computation graphs, called ES-Single, that re-uses the same outer parameter perturbations along each step of an unrolled trajectory. ES-Single has constant variance with respect to the number of partial unrolls per inner problem. Due to its low variance, ES-Single outperforms PES on a wide range of synthetic and real-world tasks. In addition, ES-Single is simpler to implement than PES, as it does not require maintaining a perturbation accumulator per particle, and only samples perturbations at the start of each inner problem, rather than for each truncated unroll.

Contributions.

- We propose an algorithm for ES-based, unbiased gradient estimation in unrolled computation graphs, called ES-Single. We motivate ES-Single by discussing its relationship to full-unroll ES and PES.
- We show that ES-Single can have substantially lower variance than PES, overcoming a key barrier for use in long-horizon inner problems, especially when using short truncated unrolls.
- We evaluate ES-Single on a diverse set of tasks, from synthetic problems designed to test unbiasedness, to hyperparameter optimization, RNN training, and meta-training learned optimizers. We found that ES-Single outperformed PES across all tasks we investigated.

We provide JAX code for ES-Single in Appendix H, and a Colab notebook implementation here.

2. Background

We follow the problem setup of Vicol et al. (2021), considering an unrolled computation graph with state $\boldsymbol{s}_t$ at time t, updated via a function f parameterized by $\boldsymbol{\theta}$:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t; \boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{x}_t$ is an optional input at each step (e.g., data). Many common tasks in machine learning are instances of this problem: for example, when training an RNN, $\boldsymbol{s}_t$ is the hidden state and $\boldsymbol{\theta}$ are the RNN parameters, while for hyperparameter optimization, $\boldsymbol{s}_t$ represents the parameters of a neural network being optimized and $\boldsymbol{\theta}$ are hyperparameters such as the learning rate and momentum. The objective function for optimizing $\boldsymbol{\theta}$ is the sum of per-timestep losses $L_t(\boldsymbol{s}_t; \boldsymbol{\theta})$:

$$L(\boldsymbol{\theta}) = \sum_{t=1}^{T} L_t(\boldsymbol{s}_t; \boldsymbol{\theta}) \tag{2}$$

Appendix A summarizes the notation used in this paper.

Chaos and Smoothing. Unrolling computation graphs can give rise to chaotic dynamics, which pose fundamental challenges for gradient-based methods (Parmas & Sugiyama, 2019). Chaos frequently arises in meta-optimization (for example hyperparameter optimization and training learned optimizers), due to nonlinear inner loss surfaces which have many local minima—small changes to the outer parameters may lead to different local minima, that have different meta-loss values. To overcome chaos, one may consider optimizing a Gaussian-smoothed outer loss, $\tilde{L}(\boldsymbol{\theta}) = \mathbb{E}_{\tilde{\boldsymbol{\theta}} \sim \mathcal{N}(\boldsymbol{\theta}, \sigma^2 \mathbf{I})} \left[L(\tilde{\boldsymbol{\theta}}) \right]$. Common approaches for computing the gradient of the smoothed objective $\tilde{L}(\boldsymbol{\theta})$ include evolution strategies and the reparameterization gradient (Ruiz et al., 2016).

Vanilla Evolution Strategies. Evolution strategies (Rechenberg, 1973; Nesterov & Spokoiny, 2017) is a method for zeroth-order gradient estimation, that computes a stochastic finite-difference estimate of the gradient as follows:

$$\begin{aligned} \boldsymbol{g}^{\text{ES}} &= \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})} \left[\boldsymbol{\epsilon} L(\boldsymbol{\theta} + \boldsymbol{\epsilon}) \right] \\ &\approx \frac{1}{\sigma^2 N} \sum_{i=1}^{N} \boldsymbol{\epsilon}^{(i)} L(\boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}) \end{aligned}$$

where N is the number of Monte Carlo samples (also called *particles*) used to estimate the expectation. Antithetic sampling (Owen, 2013) is a widely-used technique to reduce the variance of ES, that works by sampling pairs of positive and negative perturbations, $\boldsymbol{g}^{\text{ES-A}} = \mathbb{E}_{\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})} \left[\boldsymbol{\epsilon} (L(\boldsymbol{\theta} + \boldsymbol{\epsilon}) - L(\boldsymbol{\theta} - \boldsymbol{\epsilon})) \right]$. In practice, one typically uses a Monte Carlo estimate, denoted with a hat, as follows:

$$\hat{\boldsymbol{g}}^{\text{ES-A}} = \frac{1}{N \sigma^2} \sum_{i=1}^{N/2} \boldsymbol{\epsilon}^{(i)} (L(\boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}) - L(\boldsymbol{\theta} - \boldsymbol{\epsilon}^{(i)})) \tag{3}$$

where N is even, and $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. Unfortunately, applying vanilla ES to long inner problems leads to slow updates, while using partial unrolls leads to truncation bias (Metz et al., 2019).

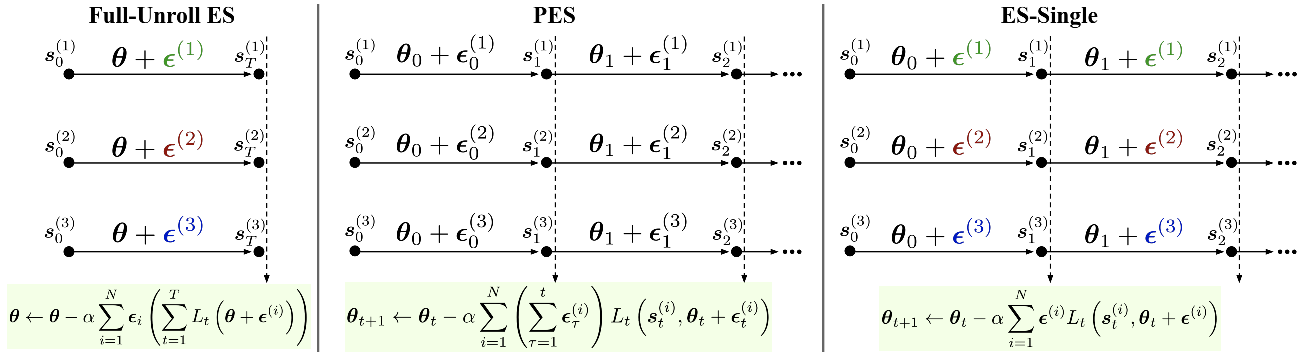

Full-Unroll ES
PES
ES-Single
$s_0^{(1)}$ $\theta + \epsilon^{(1)}$ $s_T^{(1)}$
$s_0^{(2)}$ $\theta + \epsilon^{(2)}$ $s_T^{(2)}$
$s_0^{(3)}$ $\theta + \epsilon^{(3)}$ $s_T^{(3)}$
$\theta \leftarrow \theta - \alpha \sum_{i=1}^{N} \epsilon_i \left(\sum_{t=1}^{T} L_t \left(\theta + \epsilon^{(i)} \right) \right)$
$s_0^{(i)}$ $\theta_0 + \epsilon_0^{(i)}$ $s_1^{(i)}$ $\theta_1 + \epsilon_1^{(i)}$ $s_2^{(i)}$
$\theta_{t+1} \leftarrow \theta_t - \alpha \sum_{i=1}^{N} \left(\sum_{\tau=1}^{t} \epsilon_\tau^{(i)} \right) L_t \left(s_t^{(i)}, \theta_t + \epsilon_t^{(i)} \right)$
$s_0^{(i)}$ $\theta_0 + \epsilon^{(i)}$ $s_1^{(i)}$ $\theta_1 + \epsilon^{(i)}$ $s_2^{(i)}$
$\theta_{t+1} \leftarrow \theta_t - \alpha \sum_{i=1}^{N} \epsilon^{(i)} L_t \left(s_t^{(i)}, \theta_t + \epsilon^{(i)} \right)$

Figure 2. **Comparison of the computation graphs of full-unroll ES (left), PES (middle), and ES-Single (right).** Full-Unroll ES samples a perturbation $\epsilon^{(i)}$ for particle i, and runs a full unroll from state $s_0^{(i)}$ to $s_T^{(i)}$ using perturbed parameters $\theta + \epsilon^{(i)}$. Both PES and ES-Single split the computation graph into a sequence of partial unrolls, but differ in how perturbations are sampled and how intermediate results are aggregated to update the outer parameters online: PES samples a new perturbation $\epsilon_t^{(i)}$ for particle i in each unroll t, and sums the perturbations experienced by each particle up to the current point in the inner problem, $\sum_{\tau=1}^{t} \epsilon_\tau^{(i)}$; in contrast, ES-Single samples a single perturbation $\epsilon^{(i)}$ per particle at the start of each inner problem—keeping it fixed for the duration of the problem—and does not sum perturbations over time. ES-Single can be interpreted as inserting *breakpoints into the full-unroll ES computation*, at which the intermediate losses are aggregated to form a gradient estimate used to update the outer parameters. The computation graph for vanilla truncated ES is provided in Appendix C.1.

Persistent Evolution Strategies (PES). PES (Vicol et al., 2021) is an ES-based approach for unbiased gradient estimation using partial unrolls of the inner problem. The PES gradient estimator is defined as follows, where θ_t denotes the application of the shared parameters θ at step t and the loss L_t is written explicitly as a function of all applications of θ up to the current time, rather than as a function of the state s_t that implicitly depends on past θ's:

$$g^{\text{PES}} = \frac{1}{\sigma^2} \mathbb{E}_{\epsilon} \left[\sum_{t=1}^{T} \left(\sum_{\tau=1}^{t} \epsilon_\tau \right) L_t(\theta_1 + \epsilon_1, \dots, \theta_t + \epsilon_t) \right]$$

Here, the expectation is over a $T \times P$ matrix whose rows are the per-timestep perturbations ϵ_t. Intuitively, PES maintains a collection of particles, and applies a different outer parameter perturbation for each partial unroll of the inner problem. The particles are not reset after each partial unroll, and the perturbations experienced by each particle are accumulated over the course of an inner problem. The particle states and accumulators reset at the start of a new inner problem. Vicol et al. (2021) showed that the variance of PES depends on the covariance between gradients of each loss term L_t with respect to per-timestep parameters θ_τ. They analyzed several scenarios with different covariance assumptions, and found that in a real-world scenario, the variance increases as the number of unrolls per inner problem increases (Figure 1); thus, while PES works well for tasks with an intermediate number of unrolls (e.g., 10-100 unrolls), it struggles with longer tasks due to variance.

3. ES-Single

We propose an algorithm for ES-based gradient estimation in unrolled computation graphs, that is simpler to implement than PES, has low variance, and performs well on a variety of tasks. To introduce this algorithm, we first revisit full-unroll ES, which computes $\frac{1}{\sigma^2}\mathbb{E}_{\epsilon}[\epsilon L(\theta + \epsilon)] \approx \frac{1}{N\sigma^2} \sum_{i=1}^{N} \epsilon^{(i)} L(\theta + \epsilon^{(i)})$. In full-unroll ES, we initialize N particles, sample an outer parameter perturbation $\epsilon^{(i)}$ for each particle, unroll the full inner problem using the perturbed outer parameters $\theta + \epsilon^{(i)}$, and aggregate the results to form the gradient estimate. Full-unroll ES yields useful gradient estimates, but is impractical due to the high latency between parameter updates (which is especially problematic for tasks such as hyperparameter optimization, where the inner problem typically has length $T > 10^3$). However, one can consider splitting the computation graph into a series of truncated unrolls, and using the intermediate results obtained after each unroll to update the outer parameters more frequently. The ES-Single algorithm inserts breakpoints in the inner optimization, at which the intermediate results (e.g., the losses from the current unroll) are aggregated to form a gradient estimate that is used to update the outer parameters. Mathematically, the gradient estimator for ES-Single is equivalent to the full-unroll ES gradient (shown here using antithetic sampling),

$$g^{\text{ES-Single}} = \frac{1}{\sigma^2} \mathbb{E}_{\epsilon} \left[\epsilon (L(\theta + \epsilon) - L(\theta - \epsilon)) \right], \qquad (4)$$

where $\epsilon \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$. However, ES-Single differs from full ES *algorithmically*: full ES treats the inner problem as a black box, ignoring its iterative nature; in contrast, ES-Single treats it as a *gray-box*, which leverages this structure by constructing gradient estimates from intermediate progress, and updating the outer parameters online.

ES-Single samples perturbations $\epsilon^{(i)}$ for each particle once,

Algorithm 1 Truncated Evolution Strategies (ES) applied to partial unrolls of a computation graph.

```
Input: $s_0$, initial state
       $K$, truncation length for partial unrolls
       $N$, number of particles
       $\sigma$, standard deviation of perturbations
       $\alpha$, learning rate for outer optimization
Initialize $s = s_0$
while inner problem not finished do
    $\hat{g}^{\text{ES}} \leftarrow \mathbf{0}$
    for $i = 1, \dots, N$ do
        $\epsilon^{(i)} = \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 \mathbf{I}) & i \text{ odd} \\ -\epsilon^{(i-1)} & i \text{ even} \end{cases}$
        $\hat{L}_K^{(i)} \leftarrow \text{unroll}(s, \theta + \epsilon^{(i)}, K)$
        $\hat{g}^{\text{ES}} \leftarrow \hat{g}^{\text{ES}} + \epsilon^{(i)} \hat{L}_K^{(i)}$
    end for
    $\hat{g}^{\text{ES}} \leftarrow \frac{1}{N\sigma^2} \hat{g}^{\text{ES}}$
    $s \leftarrow \text{unroll}(s, \theta, K)$
    $\theta \leftarrow \theta - \alpha \hat{g}^{\text{ES}}$
end while
```

Algorithm 2 ES with a single perturbation per particle re-applied in each truncated unroll (ES-Single).

```
Input: $s_0$, initial state
       $K$, truncation length for partial unrolls
       $N$, number of particles
       $\sigma$, standard deviation of perturbations
       $\alpha$, learning rate for outer optimization
Initialize $s^{(i)} = s_0$ for $i \in \{1, \dots, N\}$
for $i = 1, \dots, N$ do
    $\epsilon^{(i)} = \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 \mathbf{I}) & i \text{ odd} \\ -\epsilon^{(i-1)} & i \text{ even} \end{cases}$
end for
while inner problem not finished do
    $\hat{g}^{\text{ES-Single}} \leftarrow \mathbf{0}$
    for $i = 1, \dots, N$ do
        $s^{(i)}, \hat{L}_K^{(i)} \leftarrow \text{unroll}(s^{(i)}, \theta + \epsilon^{(i)}, K)$
        $\hat{g}^{\text{ES-Single}} \leftarrow \hat{g}^{\text{ES-Single}} + \epsilon^{(i)} \hat{L}_K^{(i)}$
    end for
    $\hat{g}^{\text{ES-Single}} \leftarrow \frac{1}{N\sigma^2} \hat{g}^{\text{ES-Single}}$

    $\theta \leftarrow \theta - \alpha \hat{g}^{\text{ES-Single}}$
end while
```

Figure 3. **A comparison of the vanilla truncated ES and ES-Single gradient estimators**, applied to partial unrolls of a computation graph. The conditional statement for $\epsilon^{(i)}$ is used to implement antithetic sampling. Differences between the two algorithms are highlighted in red. While ES samples different perturbations for each particle in each partial unroll, ES-Single samples one perturbation per particle before the inner problem starts, and re-applies the same perturbation in each partial unroll comprising the inner problem.

at the start of an inner problem, which are then kept fixed for the entirety of the inner problem—the same perturbations are applied to the outer parameters at each partial unroll. Because ES-Single is mathematically equivalent to full-unroll ES, it is unbiased by construction:

Proposition 3.1 (ES-Single is unbiased). *Assume that $L(\theta)$ is quadratic and $\nabla_\theta L(\theta)$ exists. Then, the ES-Single gradient estimator using antithetic sampling is unbiased, that is,* $\text{bias}(\hat{g}^{\textit{ES-Single}}) = \mathbb{E}_\epsilon \left[\hat{g}^{\textit{ES-Single}} \right] - \nabla_\theta L(\theta) = 0.$

Proof. The proof is provided in Appendix D.1. □

The main difference between ES-Single and truncated ES is that ES-Single maintains separate states for each particle throughout the full inner problem (that are updated in parallel in each partial unroll), rather than collapsing the particles to update a single mean state s after each truncated unroll. Also, ES-Single differs from PES in two key ways: 1) it uses the same perturbations over all partial unrolls of an inner problem, rather than sampling new perturbations for each partial unroll; and 2) it does not accumulate the perturbations over time, as done in PES. Thus, ES-Single is simple to implement, and may be slightly cheaper per iteration if the cost of sampling perturbations is high (e.g., for high-dimensional parameters).

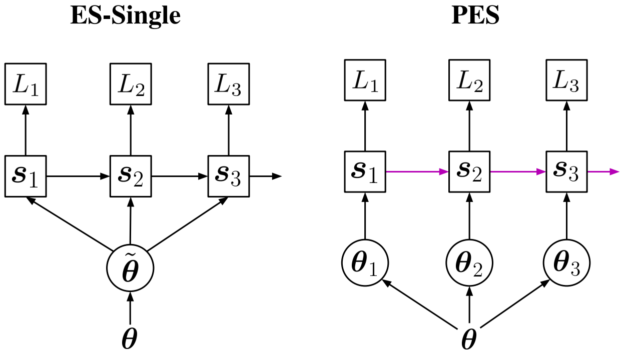


Figure 4. Stochastic computation graphs for ES-Single and PES, using the notation from Schulman et al. (2015). Vanilla truncated ES can be seen as removing the recurrent connections from PES.

Stochastic Computation Graph. Figure 4 compares the stochastic computation graphs for ES-Single to those of PES and ES. In these diagrams, squares represent deterministic nodes, which are functions of their parents; circles represent stochastic nodes (e.g., $\tilde{\theta}$) which are distributed conditionally on their parents; and nodes not in squares or circles (e.g., θ) represent inputs. ES-Single samples a single perturbed outer parameter $\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 \mathbf{I})$ that influences all the states s_t in the unroll, such that all losses L_t are downstream of the node $\tilde{\boldsymbol{\theta}}$. In contrast, PES perturbs the outer parameters independently for each partial unroll, $\boldsymbol{\theta}_t \sim \mathcal{N}(\boldsymbol{\theta}, \sigma^2 \mathbf{I})$; in this case, the losses downstream of node $\boldsymbol{\theta}_t$ are $\{L_\tau\}_{\tau=t}^T$. In Appendix E, we provide derivations of each gradient estimator, leveraging Theorem 1 from Schulman et al. (2015), which gives a generic formula for an unbiased estimator of such graphs.

Generalization of ES-Single and PES. One can also consider a generalization of both ES-Single and PES, that decouples the interval at which we update the perturbation accumulator from the interval at which we update the outer parameters. In particular, one can introduce another hyperparameter, Ω, that specifies the meta-update interval, while K denotes the interval at which new perturbations are sampled and at which the perturbation accumulator is updated. Many algorithms of interest can be obtained as special cases, by setting K and Ω appropriately. Let T be the length of a full inner problem. Then, 1) if $K = \Omega = T$, we recover full-unroll ES; 2) if $K = \Omega$ and $K < T$, we recover PES; 3) if $K = T$ and $\Omega < T$, we recover ES-Single; and 4) if $K, \Omega < T$ and $\Omega < K$, we obtain a new estimator whose properties lie between the others. The stochastic computation graph for this generalization, and the derivation of the resulting unbiased estimator, are provided in Appendix F.

3.1. Variance

In contrast to PES, the variance of the ES-Single estimator does not depend on the number of partial unrolls per inner problem. We measured the empirical variance on the same task used by Vicol et al. (2021): we consider a tiny LSTM trained on the character-level Penn TreeBank dataset (Marcus et al., 1993). The full inner problem consists of a sequence of length $T = 1000$, which we split into truncated unrolls of lengths $K \in \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000\}$. When measuring the variance of each estimator, we keep the parameters $\boldsymbol{\theta}$ fixed—that is, we do not update $\boldsymbol{\theta}$ after each partial unroll—and accumulate the gradient for the full problem by summing the estimates over the truncated unrolls. This allows us to avoid any hysteresis effects. We considered three different scenarios: 1) a sequence consisting of random characters, such that the gradients at each truncated unroll are i.i.d.; 2) a sequence consisting of a single repeated character, such that the gradients from each unroll are identical; and 3) a sequence of real data from the PTB dataset. The results are shown in Figure 1: we see that ES-Single has similar variance for all three scenarios, and in each case the variance is constant with respect to the number of unrolls, in contrast to PES. Thus, ES-Single has substantially lower variance, especially when the inner problem is split into many unrolls; however, PES does have slightly lower variance for intermediate numbers of unrolls (e.g., 10-100 unrolls per inner problem). Formally, ES-Single has the same variance characteristics as full-unroll ES.

Proposition 3.2 (ES-Single Variance)**.** *The total variance of ES-Single using antithetic sampling is $tr(Var(\hat{\boldsymbol{g}}^{\text{ES-Single}})) = (P+1)\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|^2$, where P is the dimensionality of $\boldsymbol{\theta}$.*

Proof. The proof is provided in Appendix D.2. □

The total variance normalized by the squared gradient norm is $O(P)$, growing linearly in the number of outer parameters. In contrast, the variance of PES includes terms in T, the number of unrolls per inner problem (Vicol et al., 2021); depending on the correlation between gradients at each partial unroll, its variance either decreases slightly with increasing T, or increases linearly with T. In the realistic scenario using a true sequence from the PTB dataset, the variance of PES initially decreases slightly as the number of inner unrolls increases, after which it increases linearly.

3.2. Hysteresis

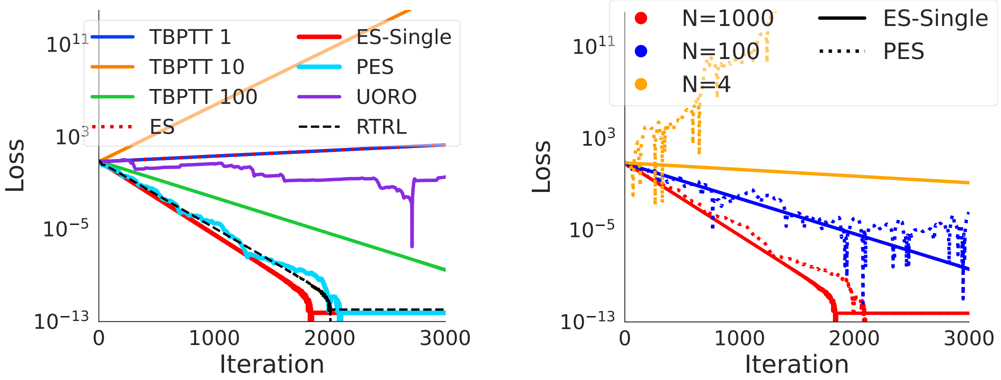
Any method that makes updates to the outer parameters online during optimization of an inner problem will suffer from *hysteresis*, including RTRL and its approximations (UORO, KF-RTRL, OK), PES, and ES-Single. The impact of hysteresis on final performance is problem-dependent; one approach to help mitigate the effects of hysteresis is to use breakstep (as opposed to lockstep) training, described in Appendix C.2.

4. Experiments

We evaluated ES-Single on several tasks from Vicol et al. (2021), which include both toy problems and real-world tasks. First, we show empirically that ES-Single is unbiased, via an influence balancing task that is designed such that truncated methods fail; then, we use ES-Single to optimize hyperparameters, to tune several mixed continuous and discrete hyperparameters for a FashionMNIST training task. Finally, we consider two high-dimensional problems: 1) training an LSTM to copy sequences of increasing length (on which truncated methods fail); and 2) meta-training a learned optimizer. Both of these tasks have thousands of outer parameters, which allows us to evaluate the scalability of ES-Single to real-world settings. Overall, we show that ES-Single is unbiased, and consistently outperforms PES on all tasks, achieving lower meta-loss values in fewer meta-optimization steps. Experimental details and additional results are provided in Appendix C.

4.1. Synthetic Influence Balancing Task

First, we revisit the influence balancing task, originally introduced by Tallec & Ollivier (2017a) and used in PES (Vicol et al., 2021). This is a synthetic task with a scalar parameter $\theta \in \mathbb{R}$, designed such that θ has a negative influence in the short term but a positive influence in the long term. We



(a) Comparing ES-Single to TBPTT, vanilla truncated ES, PES, UORO, and RTRL.

(b) Ablation over the number of particles N for PES and ES-Single.

Figure 5. Evaluating ES-Single on the synthetic influence balancing task from Tallec & Ollivier (2017a). Note that TBPTT with truncation lengths 10 and 100 moves in the wrong direction, and truncated ES exactly matches the behavior of TBPTT with $K = 1$.

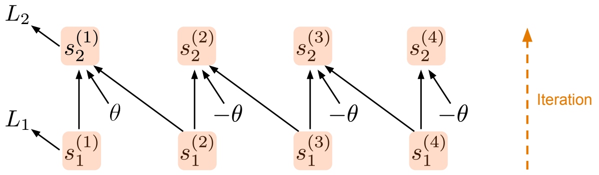


Figure 6. Illustration of the influence balancing task.

consider a linear dynamical system:

$$\boldsymbol{s}_{t+1} = \mathbf{A}\boldsymbol{s}_t + (\underbrace{\theta, \ldots, \theta}_{p \text{ positive}}, \underbrace{-\theta, \ldots, -\theta}_{n-p \text{ negative}})^\top \qquad (5)$$

where $\mathbf{A}$ is an $n \times n$ matrix with $\mathbf{A}_{i,i} = 0.5$, $\mathbf{A}_{i,i+1} = 0.5$, and 0 everywhere else. The loss L_t computes the squared error on the first index in the state vector $\boldsymbol{s}_t$; see Appendix C for details. This task is shown diagrammatically in Figure 6. As shown in Figure 5a, ES-Single outperforms PES and RTRL, and yields a smoother loss curve. Note that for this task, the inner problem is infinite; thus, the perturbation accumulator for PES is never reset. Because the variance of PES increases with the number of unrolls, PES requires a large number of particles ($N = 1000$) to perform well. If the number of particles is decreased, optimization becomes unstable, as shown in Figure 5b. In contrast, because the variance of ES-Single does not depend on the number of unrolls, it can perform well on this task with substantially fewer particles, even $N = 4$.

4.2. Hyperparameter Optimization

MNIST LR Schedule. Here, we used ES-Single to meta-learn a learning rate (LR) schedule used to train an MLP on MNIST. Based on (Wu et al., 2018), we used a two-hidden-layer MLP with 100 units per layer, and tuned LR schedule parameterized by $\alpha_t = \frac{\theta_0}{(1+\frac{t}{Q})^{\theta_1}}$, where θ_0 is the initial LR, θ_1 is the LR decay factor, and $Q = 5000$ is a constant. The results are shown in Figure 7. We found that ES-Single performed similarly to PES, but had more stable convergence near the optimum, while PES at times drifted away from the optimum due to its high variance.

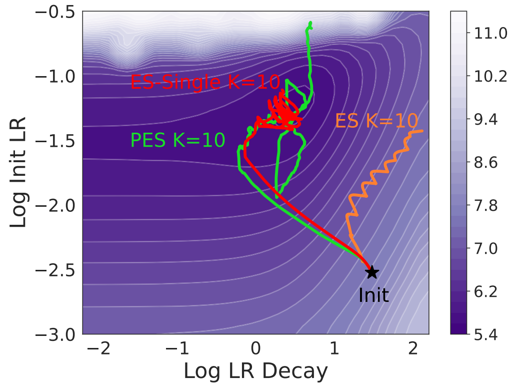
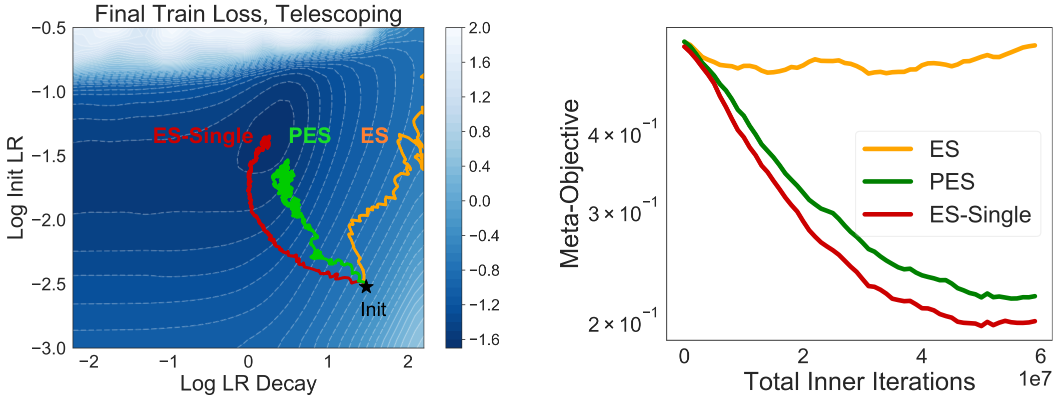


Figure 7. Meta-optimization of a learning rate schedule for an MNIST MLP, using truncation length $K = 10$. Darker regions are better. Further experiments are provided in Appendix C.

Telescoping Sums. If the desired meta-objective is the *final* loss L_T rather than the sum of losses $\sum_{t=0}^{T} L_t$, then this can be handled gracefully in our framework by defining $p_t = L_t - L_{t-1}$, where we define $L_{-1} \equiv 0$ for notational simplicity. Then, we can consider the sum of p_t, which yields a telescoping sum:

$$\sum_{t=0}^{T} p_t = (\cancel{L_0} - L_{-1}) + \cdots + (L_T - \cancel{L_{T-1}}) = L_T \qquad (6)$$

Figure 8 compares vanilla truncated ES, PES, and ES-Single on a task that tunes the learning rate and decay factor for training an MLP on FashionMNIST, targeting the final training loss. We see that the meta-optimization trajectory of ES-Single was significantly smoother than that of PES, more closely followed the meta-loss contours, and had better stability near the optimum (Figure 8a). As shown in



(a) Meta-optimization using telescoping sums.

(b) Meta-losses obtained by ES, PES, and ES-Single.

Figure 8. Meta-optimizing a learning rate schedule for an MLP on FashionMNIST, using a telescoping sum to target the final training loss.

Figure 8b, ES-Single converged more rapidly to the optimal meta-objective value than PES.

Tuning Many Hyperparameters. Here, we applied ES-Single to tune many hyperparameters simultaneously, to train a 5-hidden-layer MLP on FashionMNIST. The meta-objective is the sum of validation losses over the inner problem. We tuned 29 hyperparameters, including separate learning rates and momentum coefficients per parameter block (e.g., for each weight matrix and bias vector in the MLP), and the number of hidden units per layer (which is a discrete hyperparameter that takes values in the range 10-100). We compared ES-Single to random search, vanilla ES, and PES. The results are shown in Figure 9: we found that ES-Single substantially outperformed these baselines, and achieved lower meta-loss in fewer iterations compared to PES.

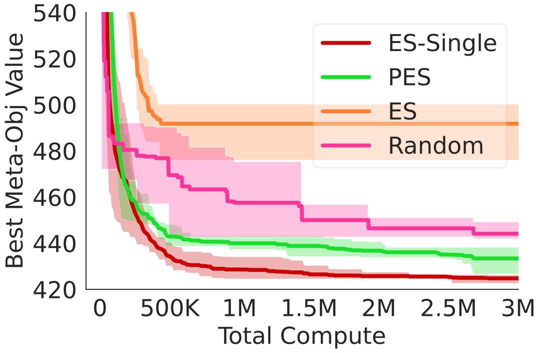


Figure 9. Tuning many hyperparameters for an MLP on FashionMNIST, targeting the sum of validation losses as the meta-objective.

4.3. LSTM Copy Task

Next, we used ES-Single to train an LSTM on the copy task introduced by Mujika et al. (2018), where the model must read a binary string of length T, and output the same string. The challenge lies in learning long-term dependencies as T increases. Following Mujika et al. (2018), we use a curriculum starting with $T = 1$, and increasing T by 1 each time the exponential moving average of the cross-entropy loss (e.g., bits-per-character) drops below the threshold 0.15. To ensure that the model does not overfit to a particular sequence length, we sample T uniformly from $\{T-5, \dots, T\}$ (or $T = 1$ if the sampled value is negative). We train a 1-layer LSTM with hidden state size 100, that has 42804 parameters, which we learn via ES-based methods, evaluating scalability. PES and ES-Single were run using truncations of length $K = 1$ for fully-online learning; for vanilla truncated ES, we used truncation lengths $K \in \{25, 50\}$. In Figure 10, we show the maximum length T that is successfully copied over the course of training using ES, PES, and ES-Single. As expected, ES plateaus, as it intrinsically cannot model dependencies across longer horizons than its truncation length. Both PES and ES-Single outperform ES, but ES-Single substantially outperforms PES, with T increasing faster and reaching higher maximum values.

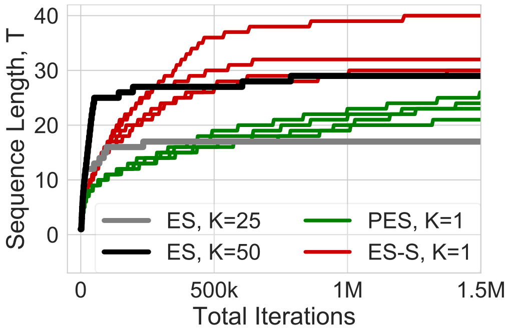


Figure 10. Maximum sequence length T that is successfully copied in the copy task from Mujika et al. (2018). Curves of the same color use different random seeds. For the ES baselines (gray and black curves), we show only the best result to reduce clutter. Here, the x-axis represents the number of tokens ingested by each approach (e.g., data-time rather than compute).

4.4. Learned Optimizer Training

Here, we used ES-Single to meta-optimize a learned optimizer using the LOLv2 architecture introduced by Metz et al. (2018). This optimizer is meta-trained to optimize a 2-hidden-layer MLP with 128 hidden units per layer, on FashionMNIST for $T = 5000$ steps, using truncated unrolls of length $K = 10$. As the meta-objective, we targeted the mean training loss over the inner optimization trajectory. In Figure 11, we show the meta-objective values obtained over the course of meta-training, using truncated ES, PES, and ES-Single. ES fails due to truncation bias, while PES performs poorly due to high variance; ES-Single performs much better on this long-horizon task with short truncations.

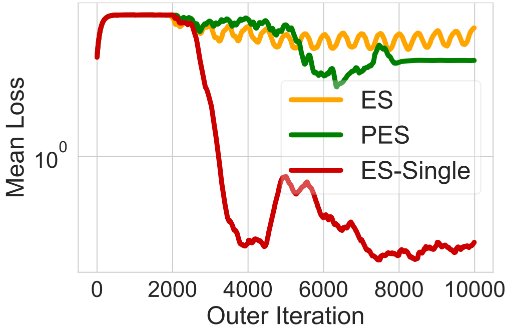


Figure 11. Meta-training a learned optimizer, targeting a two-layer MLP on FashionMNIST.

5. Related Work

We provide extended related work in Appendix B.

Gradient-Based Approaches. There are two families of gradient-based methods for computing gradients through unrolled computation, based on reverse-mode (e.g., backpropagation through time, BPTT) or forward-mode gradient accumulation (e.g., real-time recurrent learning, RTRL). Backpropagating through full unrolled sequences is expensive, with compute and memory cost that scales linearly in the unroll length. Gradient checkpointing (Chen et al., 2016) reduces the memory requirement to $O(\sqrt{T})$, at the cost of additional compute. Truncated BPTT (TBPTT) operates on shorter sub-sequences of length $K \ll T$, substantially reducing cost, but introducing truncation bias that can lead to sub-optimal solutions (Wu et al., 2018). ART-BP (Tallec & Ollivier, 2017b) uses randomly sampled truncation lengths, and introduces reweighting factors during backprop based on the sequence length to yield an unbiased gradient estimate of the total loss.

RTRL performs forward-mode gradient accumulation, by maintaining the recurrent Jacobian via the following update: $\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}} = \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_{t-1}} \frac{d\boldsymbol{s}_{t-1}}{d\boldsymbol{\theta}} + \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{\theta}}$. RTRL allows for fully online learning of the outer parameters (e.g., with outer updates taken every $K = 1$ steps), but is intractable for high-dimensional problems, as the recurrent Jacobian $\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}}$ is $P \times P$ and thus too large to store in memory. Several cheaper approximations to RTRL have been proposed, including: Unbiased Online Recurrent Optimization (UORO) (Tallec & Ollivier, 2017a), maintains a rank-1 estimate of the recurrent Jacobian; KF -RTRL (Mujika et al., 2018) proposes a Kronecker factorization of the Jacobian, and the Optimal Kronecker Sum Approximation (OK) (Benzing et al., 2019) provides a lower-variance extension of KF-RTRL. Unfortunately, these methods cannot optimize over chaotic loss landscapes, and are either high-variance, difficult to implement, or are only applicable to a restricted class of models (e.g., specific RNN architectures). Silver et al. (2021) propose a method called DODGE, for unbiased gradient estimation based on directional derivatives; being a gradient-based approach, this method requires a differentiable objective function. We provide a comparison to DODGE in Appendix C.6.

Chaos. Unrolled dynamical systems can lead to chaotic loss landscapes, for example in rigid-body physics, graphics, model-based control (Parmas et al., 2018), fluid simulation (Ni & Wang, 2017; Kochkov et al., 2021), climate modeling (Lea et al., 2000; Köhl & Willebrand, 2002), and simulation of weather (Bischof et al., 1996) or nuclear fusion (McGreivy et al., 2021). Metz et al. (2021) discuss this in depth, showing that while analytic gradients may be available in such systems, they are not necessarily useful due to high variance. In particular, the reparameterization gradient estimator (Kingma & Welling, 2013) may have orders of magnitude larger variance than black-box ES estimates (Parmas et al., 2018; Parmas & Sugiyama, 2019; Metz et al., 2019; Schwefel & Schwefel, 1977; Wierstra et al., 2014) or variational optimization (Staines & Barber, 2012). Metz et al. (2021) provide an overview of scenarios in which chaos arises, and a taxonomy of approaches to either prevent chaos from arising (e.g., switching to a better-behaved system) or to optimize in the presence of chaos (e.g., using smoothing-based approaches, as we do here). The high-level outline for ES-Single was first proposed by Vicol (2023). A paper developing a similar algorithm, written in parallel and independently from ours, is (Li et al., 2023).

6. Conclusion

We introduced an unbiased gradient estimator for unrolled computation graphs, called ES-Single. ES-Single inserts breakpoints into the computation graph for a full unroll, at which intermediate results are aggregated and used to form an ES-based gradient estimate, which is applied to update the outer parameters. Crucially, compared to vanilla truncated ES and PES, ES-Single samples outer parameter perturbations once at the start of each inner problem, and re-applies the same perturbations in each partial unroll. ES-Single is simpler to implement than PES, and has constant variance with respect to the number of partial unrolls per inner problem; this leads to substantially lower variance than PES in practice, and makes ES-Single well-suited for long-horizon tasks with short truncations. We evaluated ES-Single on a diverse set of tasks, including a synthetic task to test for unbiasedness, hyperparameter optimization, RNN training, and training of learned optimizers. On all tasks, it outperformed ES and PES.

Acknowledgements

We thank Zico Kolter and Kevin Swersky for helpful discussions.

References

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3981–3989, 2016.

Asuncion, A. and Newman, D. UCI Machine Learning Repository, 2007.

Baydin, A. G., Cornish, R., Rubio, D. M., Schmidt, M., and Wood, F. Online learning rate adaptation with hypergradient descent. *arXiv preprint arXiv:1703.04782*, 2017.

Bengio, Y. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.

Benzing, F., Gauy, M. M., Mujika, A., Martinsson, A., and Steger, A. Optimal Kronecker-sum approximation of real time recurrent learning. In *International Conference on Machine Learning*, pp. 604–613. PMLR, 2019.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2546–2554, 2011.

Bertinetto, L., Henriques, J. F., Torr, P. H., and Vedaldi, A. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

Bischof, C. H., Pusch, G. D., and Knoesel, R. Sensitivity analysis of the MM5 weather model using automatic differentiation. *Computers in Physics*, 10(6):605–612, 1996.

Blondel, M., Berthet, Q., Cuturi, M., Frostig, R., Hoyer, S., Llinares-López, F., Pedregosa, F., and Vert, J.-P. Efficient and modular implicit differentiation. *arXiv preprint arXiv:2105.15183*, 2021.

Chandra, K., Xie, A., Ragan-Kelley, J., and Meijer, E. Gradient descent: The ultimate optimizer. *Advances in Neural Information Processing Systems*, 35:8214–8225, 2022.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Domke, J. Generic methods for optimization-based modeling. In *Proceedings of Machine Learning Research*, pp. 318–326, 2012.

Finn, C., Xu, K., and Levine, S. Probabilistic model-agnostic meta-learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 31, 2018.

Finn, C. B. *Learning to learn with gradients*. University of California, Berkeley, 2018.

Foo, C.-S., Do, C. B., and Ng, A. Y. Efficient multiple hyperparameter learning for log-linear models. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 377–384, 2008.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. Forward and reverse gradient-based hyperparameter optimization. *arXiv preprint arXiv:1703.01785*, 2017.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al. Population-based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Jamieson, K. and Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics*, 2016.

Kingma, D. P. and Welling, M. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Köhl, A. and Willebrand, J. An adjoint method for the assimilation of statistical characteristics into eddy-resolving ocean models. *Tellus A: Dynamic Meteorology and Oceanography*, 54(4):406–425, 2002.

Larsen, J., Hansen, L. K., Svarer, C., and Ohlsson, M. Design and regularization of neural networks: The optimal use of a validation set. In *IEEE Signal Processing Society Workshop*, pp. 62–71, 1996.

Lea, D. J., Allen, M. R., and Haine, T. W. Sensitivity analysis of the climate of a chaotic system. *Tellus A: Dynamic Meteorology and Oceanography*, 52(5):523–532, 2000.

Li, K. and Malik, J. Learning to optimize. *arXiv preprint arXiv:1606.01885*, 2016.

Li, K. and Malik, J. Learning to optimize neural nets. *arXiv preprint arXiv:1703.00441*, 2017.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Li, O., Harrison, J., Sohl-Dickstein, J., Smith, V., and Metz, L. Noise-reuse in online evolution strategies. *arXiv preprint arXiv:2304.12180*, 2023.

Lorraine, J. and Duvenaud, D. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

Lorraine, J., Vicol, P., and Duvenaud, D. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1540–1552, 2020.

Luketina, J., Berglund, M., Greff, K., and Raiko, T. Scalable gradient-based tuning of continuous regularization hyperparameters. In *International Conference on Machine Learning (ICML)*, pp. 2952–2960, 2016.

MacKay, M., Vicol, P., Lorraine, J., Duvenaud, D., and Grosse, R. Self-Tuning Networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations (ICLR)*, 2019.

Maclaurin, D., Duvenaud, D., and Adams, R. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning (ICML)*, pp. 2113–2122, 2015.

Maheswaranathan, N., Metz, L., Tucker, G., Choi, D., and Sohl-Dickstein, J. Guided evolutionary strategies: Augmenting random search with surrogate gradients. In *International Conference on Machine Learning*, pp. 4264–4273. PMLR, 2019.

Mania, H., Guy, A., and Recht, B. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Marcus, M. P., Marcinkiewicz, M. A., and Santorini, B. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.

McGreivy, N., Hudson, S. R., and Zhu, C. Optimized finite-build stellarator coils using automatic differentiation. *Nuclear Fusion*, 61(2):026020, 2021.

Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K., and Graves, A. Practical real time recurrent learning with a sparse approximation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=q3KSThy2GwB.

Merity, S., Keskar, N. S., and Socher, R. Regularizing and optimizing LSTM language models. In *International Conference on Learning Representations (ICLR)*, 2018.

Metz, L., Maheswaranathan, N., Cheung, B., and Sohl-Dickstein, J. Meta-learning update rules for unsupervised representation learning. *arXiv preprint arXiv:1804.00222*, 2018.

Metz, L., Maheswaranathan, N., Nixon, J., Freeman, D., and Sohl-Dickstein, J. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning (ICML)*, pp. 4556–4565, 2019.

Metz, L., Maheswaranathan, N., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 2020a.

Metz, L., Maheswaranathan, N., Sun, R., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Using a thousand optimization tasks to learn hyperparameter search strategies. *arXiv preprint arXiv:2002.11887*, 2020b.

Metz, L., Freeman, C. D., Schoenholz, S. S., and Kachman, T. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

Micaelli, P. and Storkey, A. Non-greedy gradient-based hyperparameter optimization over long horizons. *arXiv preprint arXiv:2007.07869*, 2020.

Mujika, A., Meier, F., and Steger, A. Approximating real-time recurrent learning with random Kronecker factors. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 6594–6603, 2018.

Nesterov, Y. and Spokoiny, V. Random gradient-free minimization of convex functions. *Foundations of Computational Mathematics*, 17(2):527–566, 2017.

Ni, A. and Wang, Q. Sensitivity analysis on chaotic dynamical systems by Non-Intrusive Least Squares Shadowing (NILSS). *Journal of Computational Physics*, 347:56–77, 2017.

Owen, A. B. *Monte Carlo Theory, Methods and Examples*. 2013.

Parmas, P. and Sugiyama, M. A unified view of likelihood ratio and reparameterization gradients and an optimal importance sampling scheme. *arXiv preprint arXiv:1910.06419*, 2019.

Parmas, P., Rasmussen, C. E., Peters, J., and Doya, K. PIPPS: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning (ICML)*, pp. 4062–4071, 2018.

Pedregosa, F. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning (ICML)*, pp. 737–746, 2016.

Rechenberg, I. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann-Holzboog, 1973.

Ruiz, F. R., AUEB, T. R., Blei, D., et al. The generalized reparameterization gradient. *Advances in Neural Information Processing Systems*, 2016.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning internal representations by error propagation. Technical report, California University San Diego, La Jolla Institute for Cognitive Science, 1985.

Salimans, T., Ho, J., Chen, X., Sidor, S., and Sutskever, I. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Schulman, J., Heess, N., Weber, T., and Abbeel, P. Gradient estimation using stochastic computation graphs. *Advances in Neural Information Processing Systems*, 2015.

Schwefel, H.-P. and Schwefel, H.-P. *Evolutionsstrategien für die numerische Optimierung*. Springer, 1977.

Shaban, A., Cheng, C.-A., Hatch, N., and Boots, B. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1723–1732, 2019.

Silver, D., Goyal, A., Danihelka, I., Hessel, M., and van Hasselt, H. Learning by directional gradient descent. In *International Conference on Learning Representations*, 2021.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 2951–2959, 2012.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning (ICML)*, pp. 2171–2180, 2015.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Staines, J. and Barber, D. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

Swersky, K., Snoek, J., and Adams, R. P. Freeze-thaw Bayesian optimization. *arXiv preprint arXiv:1406.3896*, 2014.

Tallec, C. and Ollivier, Y. Unbiased online recurrent optimization. *arXiv preprint arXiv:1702.05043*, 2017a.

Tallec, C. and Ollivier, Y. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017b.

Vicol, P., Metz, L., and Sohl-Dickstein, J. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *International Conference on Machine Learning (ICML)*, pp. 10553–10563, 2021.

Vicol, P., Lorraine, J., Pedregosa, F., Duvenaud, D., and Grosse, R. On Implicit Bias in Overparameterized Bilevel Optimization. In *International Conference on Machine Learning (ICML)*, 2022.

Vicol, P. A. *On Bilevel Optimization without Full Unrolls: Methods and Applications*. PhD thesis, University of Toronto (Canada), 2023.

Werbos, P. J. Backpropagation through time: What it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

Wichrowska, O., Maheswaranathan, N., Hoffman, M. W., Colmenarejo, S. G., Denil, M., de Freitas, N., and Sohl-Dickstein, J. Learned optimizers that scale and generalize. *arXiv preprint arXiv:1703.04813*, 2017.

Wierstra, D., Schaul, T., Glasmachers, T., Sun, Y., Peters, J., and Schmidhuber, J. Natural evolution strategies. *The Journal of Machine Learning Research*, 15(1):949–980, 2014.

Williams, R. J. and Peng, J. An efficient gradient-based algorithm for on-line training of recurrent network trajectories. *Neural Computation*, 2(4):490–501, 1990.

Williams, R. J. and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

Wu, Y., Ren, M., Liao, R., and Grosse, R. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

Appendix

This appendix is structured as follows:

- In Section A, we provide an overview of the notation used in this paper.
- In Section B, we provide extended related work.
- In Section C, we provide experimental details and additional results.
- In Section D, we provide proofs of all statements in the main text.
- In Section E, we present derivations of the ES-Single and vanilla ES gradient estimators using the framework of stochastic computation graphs.
- In Section F, we derive a generalization of both ES-Single and PES. We provide its stochastic computation graph and resulting algorithm.
- In Section G, we derive the variance of a generalized estimator that combines a single perturbation (kept fixed over the course of an inner problem)—as in ES-Single—with independent perturbations sampled in each partial unroll—as in PES.
- In Section H, we provide a JAX implementation of ES-Single.

A. Notation

Table 1 summarizes the notation used in this paper.

Symbol	Meaning
ES	Evolution strategies
PES	Persistent evolution strategies
ES-Single	Evolution strategies with a single perturbation re-used across unrolls
(T)BPTT	(Truncated) backpropagation through time
RTRL	Real time recurrent learning
UORO	Unbiased online recurrent optimization
T	The total sequence length / total unroll length of the inner problem
K	The truncation length for subsequences / partial unrolls
S	The dimensionality of the state of the unrolled system, dim$(\boldsymbol{s})$
P	The dimensionality of the parameters of the unrolled system, dim$(\boldsymbol{\theta})$
$\boldsymbol{\theta}$	The parameters of the unrolled system
$\boldsymbol{\theta}_t$	The parameters of the unrolled system at time t, where $\boldsymbol{\theta}_t = \boldsymbol{\theta}, \forall t$
$\boldsymbol{s}_t$	The state of the unrolled system at time t
$\boldsymbol{x}_t$	The (optional) external input to the unrolled system at time t
f	The update function that evolves the unrolled system
N	The number of particles for ES and PES
σ^2	The variance of the ES/PES perturbations
$\boldsymbol{\epsilon}_t$	A perturbation applied to the parameters $\boldsymbol{\theta}$ at timestep t
$\boldsymbol{\xi}_t$	The sum of PES perturbations up to time t, $\boldsymbol{\xi}_t = \boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_t$
Θ	A matrix whose rows are per-timestep parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T$
$L_t(\Theta)$	The loss at timestep t, $L_t(\Theta) = L_t(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t)$
$L(\boldsymbol{\theta})$, $L(\Theta)$	The total loss, $L(\boldsymbol{\theta}) = L(\Theta) = \sum_{t=1}^{T} L_t(\Theta) = \sum_{t=1}^{T} L_t(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t)$
$\boldsymbol{g}_t$	The true gradient at step t: $\nabla_{\boldsymbol{\theta}} L_t(\boldsymbol{\theta})$
$\hat{\boldsymbol{g}}^{\text{ES}}$	The vanilla ES gradient estimate (with Monte-Carlo sampling)
$\hat{\boldsymbol{g}}^{\text{ES-A}}$	The vanilla ES gradient estimate, using antithetic sampling
$\hat{\boldsymbol{g}}^{\text{PES}}$	The PES gradient estimate (with Monte-Carlo sampling)
$\hat{\boldsymbol{g}}^{\text{ES-Single}}$	The ES-Single gradient estimate (with Monte Carlo sampling)
α	The learning rate for the parameters $\boldsymbol{\theta}$
unroll$(\boldsymbol{s}, \boldsymbol{\theta}, K)$	A function that unrolls the system for K steps starting with state $\boldsymbol{s}$, using parameters $\boldsymbol{\theta}$. Returns the updated state and loss resulting from the unroll

Table 1. **Table of notation, defining the terms we use in this paper.**

B. Extended Related Work

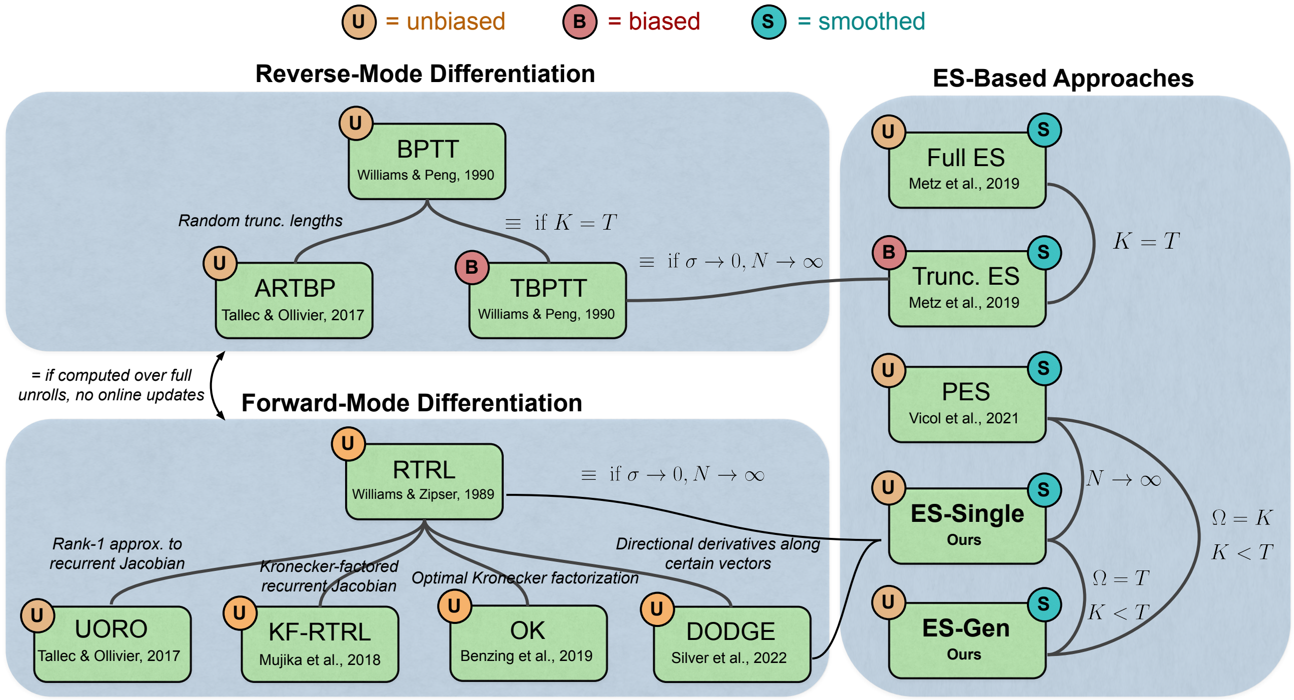

U = unbiased
B = biased
S = smoothed
Reverse-Mode Differentiation
U BPTT Williams & Peng, 1990
Random trunc. lengths
$\equiv$ if $K = T$
U ARTBP Tallec & Ollivier, 2017
B TBPTT Williams & Peng, 1990
$\equiv$ if $\sigma \to 0, N \to \infty$
ES-Based Approaches
U S Full ES Metz et al., 2019
B S Trunc. ES Metz et al., 2019
$K = T$
U S PES Vicol et al., 2021
= if computed over full unrolls, no online updates
Forward-Mode Differentiation
U RTRL Williams & Zipser, 1989
$\equiv$ if $\sigma \to 0, N \to \infty$
U S ES-Single Ours
$N \to \infty$
$\Omega = K$
$K < T$
Rank-1 approx. to recurrent Jacobian
Kronecker-factored recurrent Jacobian
Directional derivatives along certain vectors
Optimal Kronecker factorization
$\Omega = T$
$K < T$
U UORO Tallec & Ollivier, 2017
U KF-RTRL Mujika et al., 2018
U OK Benzing et al., 2019
U DODGE Silver et al., 2022
U S ES-Gen Ours

Figure 12. Connections between approaches for computing gradients in unrolled computation graphs, focusing on three categories of methods: 1) forward-mode differentiation, which includes RTRL (Williams & Zipser, 1989) and its approximations (UORO (Tallec & Ollivier, 2017a), KF-RTRL (Mujika et al., 2018), OK (Benzing et al., 2019), DODGE (Silver et al., 2021)); 2) reverse-mode differentiation, which includes backpropagation through time (BPTT), truncated BPTT, and ARTBP (Tallec & Ollivier, 2017b); and 3) evolution strategies (ES)-based approaches, which include full-unroll and truncated ES (Metz et al., 2019), PES (Vicol et al., 2021), and the generalization we introduce in Section F, which has as special cases PES and ES-Single.

Approaches for Gradient Estimation. Figure 12 illustrates connections between forward-mode, reverse-mode, and evolution strategies-based approaches to gradient estimation in unrolled computation graphs.

Black-Box, Gray-Box, and Gradient-Based Approaches. Black-box approaches to meta-optimization include random search (Bergstra et al., 2011), Bayesian optimization (Snoek et al., 2012; 2015), and full-unroll ES (Metz et al., 2019). Gray-box approaches make use of the iterative nature of the inner problem, to make faster progress than black-box methods; such approaches include Freeze-Thaw Bayesian optimization (Swersky et al., 2014), Hyperband (Li et al., 2017), Successive Halving (Jamieson & Talwalkar, 2016), Population-Based Training (Jaderberg et al., 2017), PES (Vicol et al., 2021), and ES-Single. Gradient-based approaches either: 1) differentiate through inner unrolls (Domke, 2012; Maclaurin et al., 2015; Shaban et al., 2019); 2) leverage implicit differentiation (Larsen et al., 1996; Bengio, 2000; Foo et al., 2008; Pedregosa, 2016; Luketina et al., 2016; Vicol et al., 2022; Lorraine et al., 2020; Blondel et al., 2021); or 3) leverage hypernetworks (Lorraine & Duvenaud, 2018; MacKay et al., 2019). There have also been attempts to use forward-mode gradient accumulation for hyperparameter optimization (Franceschi et al., 2017), which is only tractable when the hyperparameter dimensionality is very small (e.g., < 10). Most gradient-based approaches perform online, joint optimization over the model parameters and hyperparameters; a notable exception is Micaelli & Storkey (2020), that performs offline updates after each full inner optimization run. Black-box approaches typically do not scale well beyond ~ 10 hyperparameters. While gradient-based approaches are highly scalable, they often suffer from truncation bias, and are typically not applicable to discrete or stochastic hyperparameters (e.g., architectural hyperparameters such as the number of units per layer, or dropout rates). ES-Single is applicable to a broad range of hyperparameters, including continuous, discrete, or stochastic (e.g., dropout (Srivastava et al., 2014)) hyperparameters. In addition, it can target non-differentiable meta-objectives, such as accuracy rather than loss.

Compute and Memory Cost. Table 2 is an extension of Table 1 from Vicol et al. (2021), including an additional row for ES-Single. The compute cost of ES-Single is identical to that of PES. Similarly to PES, ES-Single maintains the states of N particles, with memory cost NS. However, ES-Single does not need to store perturbation accumulators. If the perturbations used by each particle (over the course of all unrolls in an inner problem) are sampled once at the start of the inner problem and stored in memory, then this would require NP memory (similarly to the perturbation accumulators). But the perturbations do not need to be stored this way, as they can be re-sampled using the same random seed in each partial unroll. Thus, depending on the implementation, ES-Single has memory cost less than or equal to PES.

Table 2. **Comparison of approaches for learning parameters in unrolled computation graphs**. S is the size of the system state (e.g. the RNN hidden state dimension, or in the case of hyperparameter optimization the inner-problem's weight dimensionality and potentially the optimizer state; P is the dimensionality of $\boldsymbol{\theta}$; T is the total number of steps in a sequence/unroll; K is the truncation length; and N is the number of samples (also called *particles*) used for the reparameterization gradient and in ES-based algorithms; F and B are the costs of a forward and backward pass, respectively; terms in red denote computation/memory that can be split across parallel workers.

Method	Compute	Memory	Parallel	Unbiased	Optimize Non-Diff.	Smoothed
BPTT (Rumelhart et al., 1985)	$T(F+B)$	TS	✗	✓	✗	✗
TBPTT (Williams & Peng, 1990)	$K(F+B)$	KS	✗	✗	✗	✗
ARTBP (Tallec & Ollivier, 2017b)	$K(F+B)$	KS	✗	✓	✗	✗
RTRL (Williams & Zipser, 1989)	$PS^2 + S(F+B)$	$SP + S^2$	✗	✓	✗	✗
UORO (Tallec & Ollivier, 2017a)	$F + B + S^2 + P$	$S + P$	✗	✓	✗	✗
Reparam. (Metz et al., 2019)	$NT(F+B)$	NTS	✓	✓	✗	✓
ES (Rechenberg, 1973)	NTF	NS	✓	✓	✓	✓
Trunc. ES (Metz et al., 2019)	NKF	NS	✓	✗	✓	✓
PES (Vicol et al., 2021)	NKF	$N(S+P)$	✓	✓	✓	✓
ES-Single (Ours)	NKF	$N(S+P)$	✓	✓	✓	✓

C. Experimental Details and Additional Results

In this section, we provide experimental details and additional results comparing ES-Single to truncated ES and PES. For all approaches (vanilla ES, PES, and ES-Single), we use antithetic sampling.

C.1. Truncated ES

Figure 13 shows the computation graph for vanilla truncated ES, to illustrate how it differs from full-unroll ES, PES, and ES-Single as shown in Figure 2.

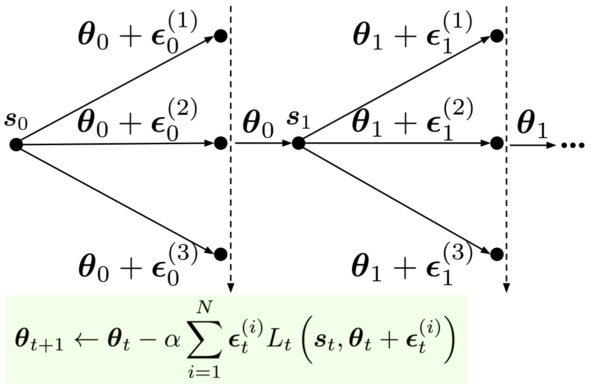
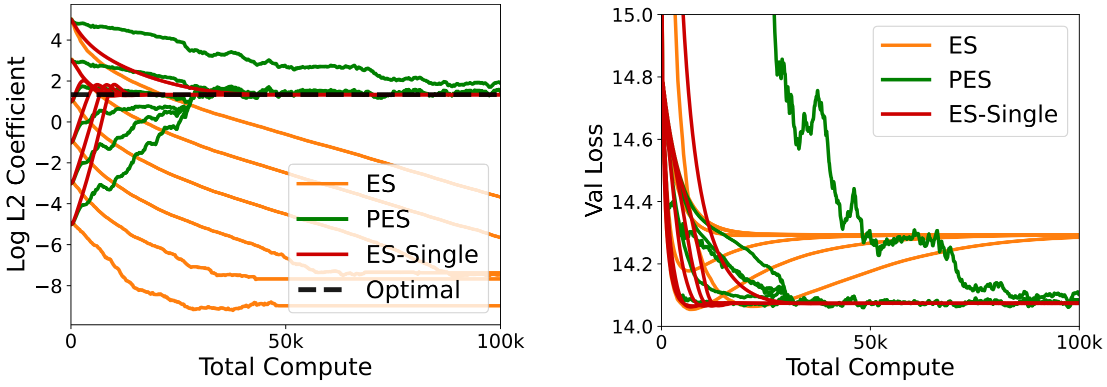

$\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}_0^{(1)}$
$\boldsymbol{\theta}_1 + \boldsymbol{\epsilon}_1^{(1)}$
$\boldsymbol{s}_0$
$\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}_0^{(2)}$
$\boldsymbol{\theta}_0$
$\boldsymbol{s}_1$
$\boldsymbol{\theta}_1 + \boldsymbol{\epsilon}_1^{(2)}$
$\boldsymbol{\theta}_1$
...
$\boldsymbol{\theta}_0 + \boldsymbol{\epsilon}_0^{(3)}$
$\boldsymbol{\theta}_1 + \boldsymbol{\epsilon}_1^{(3)}$
$\boldsymbol{\theta}_{t+1} \leftarrow \boldsymbol{\theta}_t - \alpha \sum_{i=1}^{N} \boldsymbol{\epsilon}_t^{(i)} L_t \left(\boldsymbol{s}_t, \boldsymbol{\theta}_t + \boldsymbol{\epsilon}_t^{(i)} \right)$

Figure 13. Computation graph for vanilla truncated ES. Note that truncated ES may be applied in two different ways. In the first approach, a single state $\boldsymbol{s}_t$ is maintained at time t, which serves as the common initialization for evaluating N outer parameter perturbations $\{\boldsymbol{\theta}_t + \boldsymbol{\epsilon}_t^{(i)}\}_{i=1}^{N}$. The losses obtained from these partial unrolls are aggregated to form a gradient estimate used to update $\boldsymbol{\theta}_t \rightarrow \boldsymbol{\theta}_{t+1}$. After each partial unroll, the states resulting from the N perturbations are discarded, and the single state $\boldsymbol{s}_t$ is unrolled using the mean parameters $\boldsymbol{\theta}_t$, yielding the new initialization $\boldsymbol{s}_{t+1}$ for the subsequent unroll. In the second approach, separate states are maintained for each particle over the course of meta-optimization, and different random perturbations are used to unroll those states in each partial unroll. Both approaches suffer from truncation bias.



(a) Meta-optimization trajectories for ES, PES, and ES-Single, starting from different initializations, $\{-5, -3, -1, 1, 3, 5\}$ in log-space.

(b) Validation losses attained by each method over the course of meta-optimization.

Figure 14. Comparing meta-optimization trajectories and validation losses obtained by ES, PES, and ES-Single when tuning a global L_2 regularization coefficient for linear regression on the UCI Yacht dataset.

Hyperparameter Optimization for UCI Regression. We also revisited the UCI linear regression task used in Vicol et al. (2021), which demonstrates that truncation bias can also affect regularization hyperparameters (not only optimization hyperparameters). In this task, we tune a global L_2 regularization coefficient for linear regression on the UCI Yacht dataset (Asuncion & Newman, 2007); the training set for this dataset is small, and thus strong regularization is necessary to obtain good validation performance. In Figure 14a, we plot the optimal log L_2 coefficient obtained via a fine-grained grid search (dashed black line), and compare the meta-optimization trajectories of ES, PES, and ES-Single. In Figure 14b, we show the corresponding validation losses attained by each method. In this task, the inner problem has an infinite horizon; it is never reset. We found that ES-Single converged to the optimal L_2 value more rapidly and stably than PES. All methods used Adam with learning rate 0.003 for outer optimization, $\sigma = 0.01$, and $N = 4$ particles.

Influence Balancing. Written out, the dynamical system update $\boldsymbol{s}_{t+1} = \mathbf{A}\boldsymbol{s}_t + \boldsymbol{\theta}$ is:

$$\begin{bmatrix} s_{t+1}^{(1)} \\ s_{t+1}^{(2)} \\ \vdots \\ s_{t+1}^{(n)} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{2} \end{bmatrix} \begin{bmatrix} s_t^{(1)} \\ s_t^{(2)} \\ \vdots \\ s_t^{(n)} \end{bmatrix} + \begin{bmatrix} \theta \\ \vdots \\ \theta \\ -\theta \\ \vdots \\ -\theta \end{bmatrix} \tag{7}$$

The loss L_t computes the squared error on the first index in the state vector $\boldsymbol{s}_t$:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{t=1}^{T} L_t(\boldsymbol{\theta}) = \sum_{t=1}^{T} \frac{1}{2}\left(s_t^{(1)} - 1\right)^2 \tag{8}$$

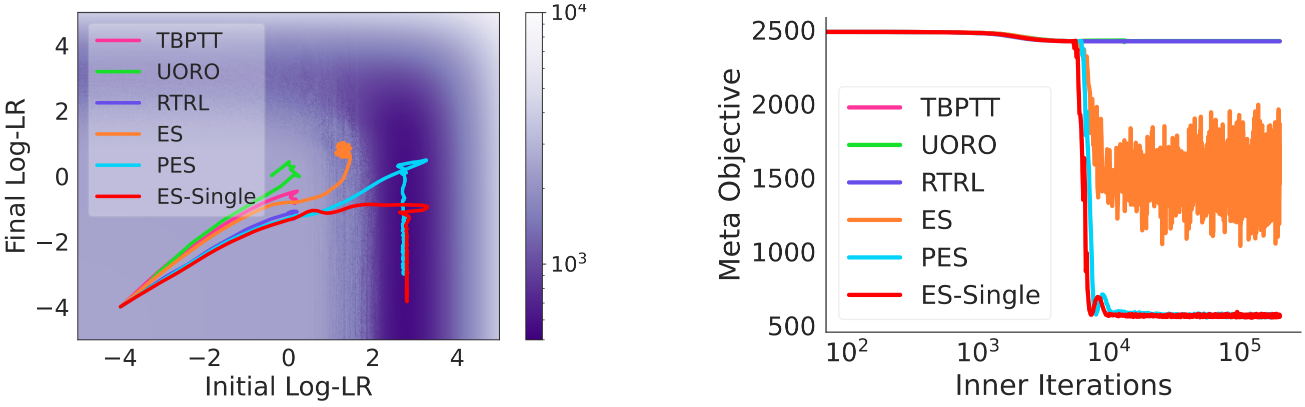
In our experiments, we used a state $\boldsymbol{s}_t$ of dimension $n = 23$, and used $p = 10$ positive copies of the scalar parameter θ concatenated with $n - p = 13$ negative copies. We initialized the state to a vector of ones, $\boldsymbol{s}_0 = \mathbf{1}$, and we inittialized $\theta = 0.5$.

Toy 2D Regression. Here, we evaluated ES-Single on a synthetic 2D task introduced by Vicol et al. (2021), which aims to learn a linearly-decaying learning rate schedule for a regression problem. The inner problem is designed to have a single global optimum but many local optima, such that small changes in the learning rate schedule can lead to convergence to different local minima; this yields a chaotic meta-loss landscape, and makes the task challenging for gradient-based outer optimizers. The learning rate at iteration t is parameterized by $\alpha_t = (1 - \frac{t}{T})e^{\theta_0} + \frac{t}{T}e^{\theta_1}$.

The inner problem involves optimizing parameters $\boldsymbol{x} = (x_0, x_1)$ to minimize the following objective function:

$$f(x_0, x_1) = \sqrt{x_0^2 + 5} - \sqrt{5} + \sin^2(x_1)\exp(-5x_0^2) + 0.25|x_1 - 100| \tag{9}$$

We used total inner problem length $T = 100$ and truncations of length $K = 10$. For all ES-based methods, we used $N = 100$ particles. For vanilla truncated ES, we used perturbation scale $\sigma = 1$, while for PES and ES-Single, we used perturbation scale $\sigma = 0.3$. For all methods, we performed outer optimization using Adam with learning rate 0.01. The results are shown in Figure 15. We found that ES-Single performed similarly to PES, both finding the optimal region of the meta-loss landscape, while the gradient-based methods (TBPTT, UORO, and RTRL) failed due to chaos in the meta-loss, and while ES failed due to truncation bias.



(a) Meta-optimization trajectories.

(b) Meta-objective values attained by each algorithm.

Figure 15. Toy regression problem, heatmap with meta-optimization trajectories overlaid, and meta-objective values over the course of training. Darker regions represent lower meta-objective values.

LSTM Copy Task. We train on minibatches of size 32, and feed the ES gradient estimates into Adam with default parameters $\beta_1 = 0.9, \beta_2 = 0.999$; for each method, we performed a grid search over learning rates $\alpha \in \{0.01, 0.001, 0.0001\}$ and perturbation scales $\sigma \in \{0.1, 0.01, 0.001, 0.0001\}$, choosing the best values based on final training performance. We used $N = 1000$ particles for each method.

As an additional result, in Figure 16, we compare PES and ES-Single to truncated backpropagation through time (denoted by TBP in the legend). Similarly to truncated ES in Figure 10, TBP also plateaus for each truncation length, as it is intrinsically limited with respect to the horizon that it can memorize.

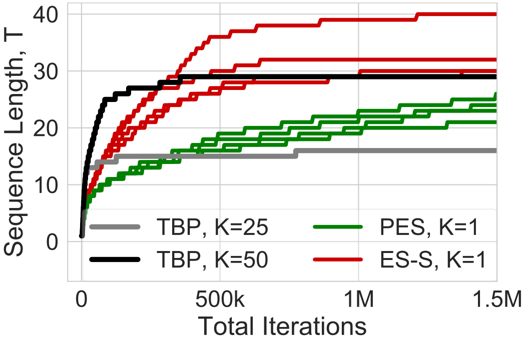


Figure 16. Maximum sequence length T that is successfully copied in the copy task from Mujika et al. (2018). Curves of the same color use different random seeds. For the truncated backprop through time (TBP) baselines (gray and black curves), we show only the best result to reduce clutter. Here, the x-axis represents the number of tokens ingested by each approach (e.g., data-time rather than compute).

Meta-Learning MNIST LR Schedule. We used a two-hidden-layer MLP with 100 hidden units per layer and ReLU activations. The learning rate schedule we meta-learn is applied to SGD with momentum, using a fixed momentum coefficient of 0.9. The total inner problem length is $T = 5000$, which is split into 500 partial unrolls of length $K = 10$. We used $N = 1000$ particles and $\sigma = 0.1$ for each estimator, and we used Adam with learning rate 1e-2 for outer optimization.

When using PES, there is a trade-off between stability and convergence speed; using a large learning rate may yield fast progress, but may lead to unstable convergence, where meta-optimization diverges away from the optimum, as shown in Figure 17. Reducing the learning rate may avoid such unstable behavior, but leads to much slower progress compared to ES-Single, as shown in Figure 18. In this experiment, ES-Single was more stable using larger learning rates than PES.

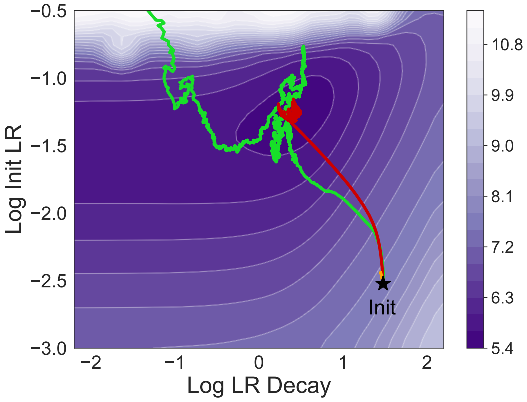


Figure 17. Meta-optimization trajectories of PES and ES-Single using truncations of length $K = 1$ on an inner problem of length $T = 5000$. With outer learning rate 1e-2, ES-Single performs well and converges stably to the optimum, while PES explodes due to variance. The red curve denotes ES-Single, while the green curve denotes PES.

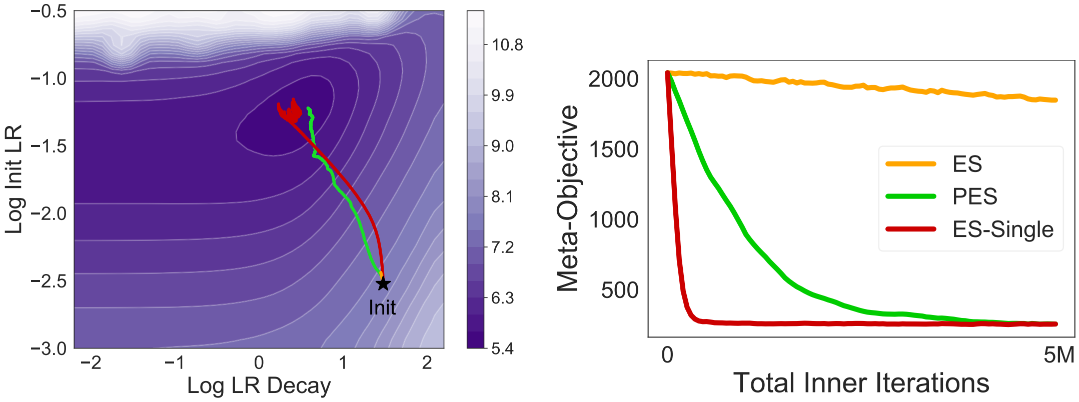
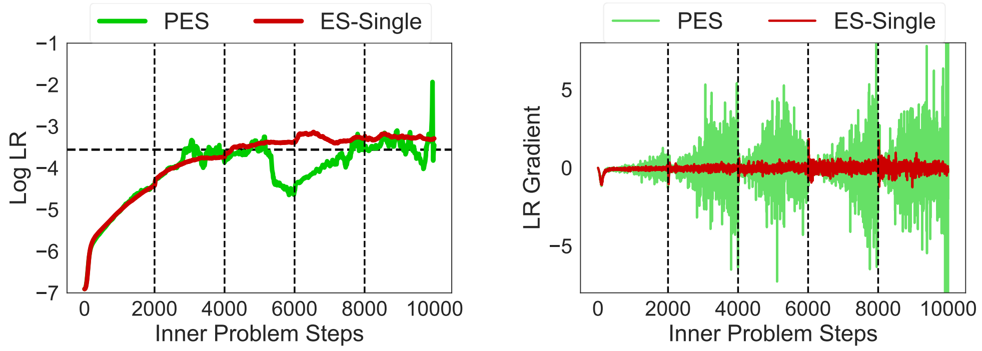


Figure 18. Meta-optimization trajectories of ES, PES, and ES-Single using truncations of length $K = 1$ on an inner problem of length $T = 5000$. The ES curve makes very little progress, so is obscured by the others. When using a smaller outer learning rate of 1e-3 for PES (to prevent it from exploding), it converges stably, but slowly.

Tuning Many Hyperparameters. Following Vicol et al. (2021), we trained an MLP with 5 hidden layers and ReLU activations on FashionMNIST, using the sum of validation losses along the inner optimization trajectory as the meta-objective. The total inner problem length was $T = 1000$, and we used truncations of length $K = 10$, yielding 100 partial unrolls per inner problem. For ES, PES, and ES-Single, we used perturbation scale $\sigma = 0.3$ and $N = 10$ particles. We used Adam as the outer optimizer, with learning rate 1e-2. The inner problem used SGD with momentum as the inner optimizer, and trained on minibatches of size 100. We tuned 29 hyperparameters in total, consisting of: a separate learning rate and momentum coefficient for each weight matrix and bias vector in the MLP (yielding 2 hyperparameters for each of the 6 weight matrices and 6 bias vectors, for 24 hyperparameters); and the number of hidden units per layer, which yields 5 additional hyperparameters. To tune the number of hidden units per layer, we used a nested dropout scheme, where the hyperparameter specifies the fraction of the maximum number of units that should be used. We set the maximum number of hidden units to 100 in each layer. All hyperparameters are optimized in the unconstrained space; each one is mapped to a



(a) Adaptation of the log-learning rate.

(b) PES and ES-Single meta-gradients over the course of multiple inner problems.

Figure 19. Comparing meta-optimization performance and meta-gradients from PES and ES-Single on a simple hyperparameter optimization task, where we aim to tune a global learning rate for training an MLP on MNIST.

constrained space by a hyperparameter-specific transformation. For learning rates, we used an exponential mapping; for momentum coefficients, we used the logistic sigmoid (such that the momentum is constrained within $(0, 1)$); and for the number of hidden units per layer, we used the sigmoid to map the unconstrained parameterization to $(0, 1)$, which represents the structured dropout rate.

For ES, PES, and ES-Single, we initialized the hyperparameters randomly: the learning rates were initialized uniformly at random in log-space, in the range $(1e - 4, 1e - 2)$; the momentum coefficients were initialized uniformly at random in logit space corresponding to the sigmoid-transformed range $(0.01, 0.9)$; and the number of hidden units is initialized randomly in logit-space corresponding to the sigmoid-transformed range $(0.2, 0.8)$. In Figure 9, we track the best meta-objective value obtained so far over the course of meta-optimization. We measure the performance as a function of total compute, which takes into account the total number of inner iterations performed (considering the number of particles used). We ran each method four times using different random seeds, and plot the mean performance as well as the min and max shown via shaded regions.

Telescoping Sums. We trained a 2-layer MLP with 100 hidden units per layer. As a computationally tractable proxy for the loss on the full training set, we sample a minibatch of size 1000 at the start of each inner problem, which is kept fixed for the duration of the problem—for telescoping sums, we evaluate the loss on the same minibatch after each partial unroll, as opposed to sampling a different random minibatch in each step, as we do when targeting the sum of losses.

Meta-Gradient Comparison. Here, we illustrate how the difference in variance between ES-Single and PES manifests in a simple hyperparameter optimization task. We tune a global learning rate used to train an MLP on MNIST. Because the outer parameter is 1-dimensional, we can find the global optimum via a fine-grained grid search, and visualize the meta-optimization iterates and meta-gradients of each algorithm. Figure 19a compares the learning rates adapted by PES and ES-Single over the course of meta-optimization; the dashed vertical lines indicate the start of each new inner problem (training is performed in lock-step, where all particles progress through the inner problem at identical iterates). We found that ES-Single converged stably towards the optimal solution, while PES was less stable due to its high variance. In Figure 19b, we show the gradient estimates produced by PES and ES-Single: we observe that the PES gradient exhibits increasingly large fluctuations over the course of each inner problem, while the ES-Single gradient is more stable. For this task, we used total inner problem length $T = 5000$, partial unrolls of length $K = 10$, and $N = 10$ particles. For each estimator, we performed a grid search over the outer learning rate and perturbation scale, choosing the best values based on convergence speed and final performance.

C.2. Lockstep vs Breakstep Training

Vanilla truncated ES, PES, and ES-Single all aggregate information across a collection of particles. In general, for such methods, there are two approaches for initializing and unrolling the particles: 1) *lockstep training*, where all particles are initialized identically, at step $t = 0$ of the inner problem, and progress through the inner optimization synchronously; or 2) *breakstep* training, in which each particle pair (considering antithetic sampling) is initialized separately, potentially at an arbitrary starting step t of the inner problem, and where particles progress through the inner problem asynchronously (e.g., one particle pair may be unrolled from $t = K$ to $t = 2K$ while another pair is unrolled from $t = 4K$ to $t = 5K$). These two approaches are illustrated in Figure 20. Often, both approaches work well; most of the experiments in this paper use lockstep training, except the learned optimizer experiment in Section 4.4, which uses breakstep training.

C.3. Effect of Smoothing

Meta-learning tasks such as hyperparameter optimization and learned optimizer training often lead to chaotic meta-loss landscapes, that are not amenable to optimization via gradient-based methods. Here, we performed an ablation over the perturbation scale σ (that controls the degree of smoothing) for ES-Single, used to optimize an MLP learned optimizer, similarly to Section 4.4 in the paper. Small perturbation scales lead to behavior similar to gradient-based methods, which may get stuck in sub-optimal local minima in chaotic loss landscapes. As shown in Figure 21, when the perturbation scale is too small, $\sigma =$ 1e-6, meta-optimization fails to make progress; in contrast, using an appropriate scale $\sigma =$ 1e-2 leads to stable convergence.

C.4. Reinforcement Learning

While investigating truncated ES-based methods for RL is an area for future work, we provide a proof-of-concept experiment here. We ran ES-Single on the continuous control task used in PES, which trains linear policy on the Swimmer MuJoCo environment. We compared vanilla truncated ES, PES, and ES-Single, all using partial unrolls of length $K = 100$. The results are shown in Figure 22, where the shaded regions denote the standard deviations over 6 random seeds. We found that ES-Single slightly outperformed PES, with smaller standard deviation, and more stable convergence to the optimal episode return.

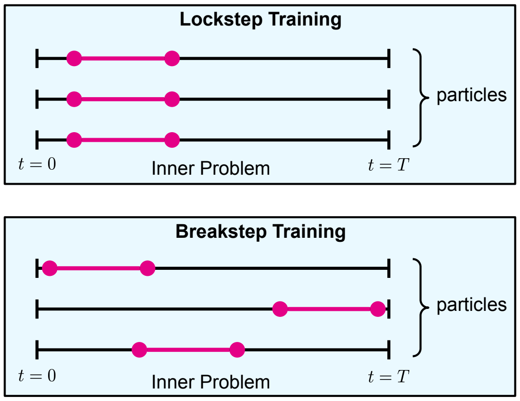


Figure 20. Conceptual illustration of lockstep and breakstep training, for methods that aggregate information across a collection of particles.

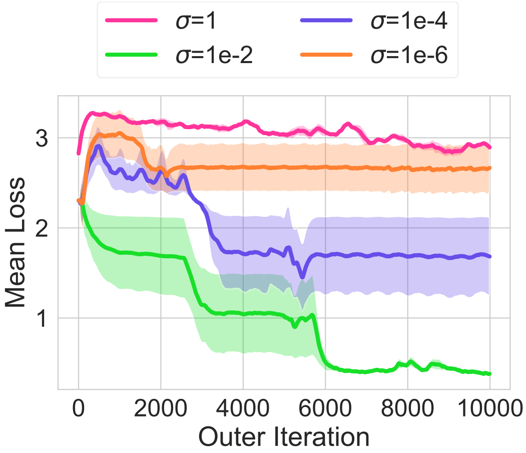


Figure 21. Ablation over the perturbation scale σ used to train a learned optimizer, targeting an MLP on FashionMNIST.

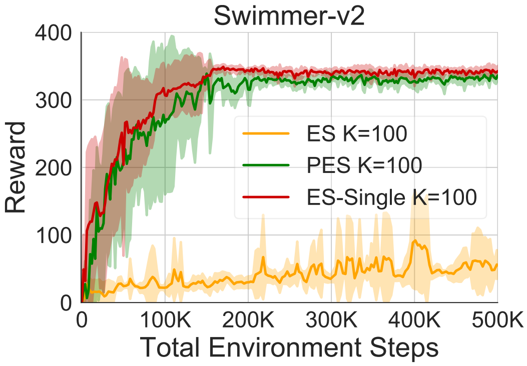
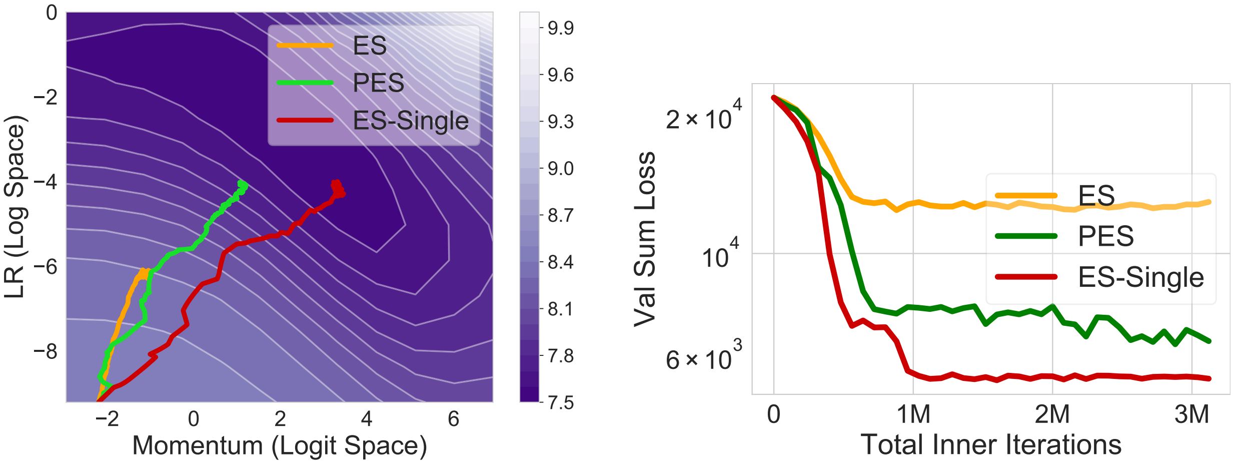


Figure 22. Learning a linear policy for the Swimmer MuJoCo environment using vanilla truncated ES, PES, and ES-Single. Here, $T = 1000$ and $K = 100$. Shaded regions denote standard deviations over 6 random seeds.



(a) Meta-optimization trajectories.

(b) Meta-objective values.

Figure 23. Tuning the learning rate and momentum coefficient for SGDm, used to optimize a ResNet on CIFAR-10. Here, $T = 5000$ and $K = 20$.

C.5. CIFAR-10 Experiments

Here, we applied ES-Single to two hyperparameter optimization tasks in which we train a ResNet on CIFAR-10. We used a 1.6M parameter Myrtle.ai ResNet architecture. In both cases, the meta-objective is the sum of validation losses over the inner problem. First, we tuned the global learning rate and momentum coefficient for SGDm. Here, the inner problem had length $T = 5000$, and we used truncations of length $K = 20$ for all approaches. As shown in Figure 23, ES-Single converged to the optimal region substantially faster than PES. Second, we tuned 24 continuous and discrete hyperparameters simultaneously. In particular, we tuned per-parameter-block learning rates and momentum coefficients, as well as the number of channels per convolutional layer. Here, the inner problem had length $T = 2000$, and we used truncations of length $K = 10$ for all approaches. The results are shown in Figure 24; we found that ES-Single substantially outperformed ES and PES, achieving lower meta-objective values using less total compute.

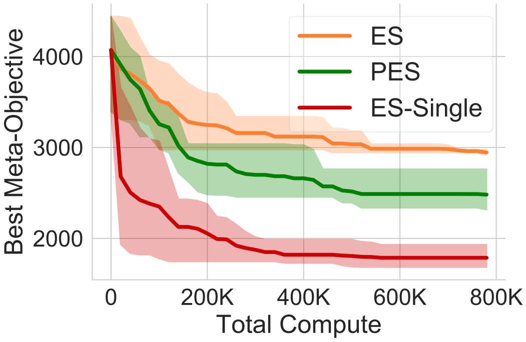


Figure 24. Tuning (continuous) per-parameter block learning rates and momentum coefficients, as well as the (discrete) number of channels per convolutional layer in a ResNet trained on CIFAR-10. The inner problem has length $T = 2000$ and we use truncations of length $K = 10$. Shaded regions denote the min/max performance over 3 random seeds.

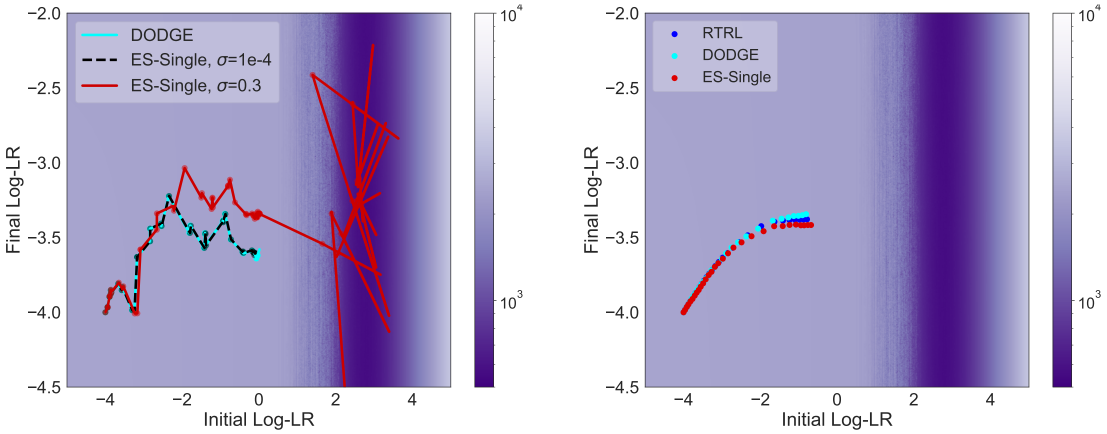
C.6. Comparison to DODGE

Here, we provide an extended discussion on the similarities and differences between ES-Single and DODGE (Silver et al., 2021). Like ES-Single, DODGE is a method for computing gradients in unrolled computation graphs. DODGE is an approximation of RTRL: rather than maintaining the expensive recurrent Jacobian matrix $\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}}$ over time, it takes the directional derivative $(\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \cdot \mathbf{u})\mathbf{u}$ along a specific vector $\mathbf{u}$. This reduces the space complexity of the algorithm, because it only needs to store and propagate a vector $\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}}\mathbf{u}$:

$$\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}}\mathbf{u} = \sum_{t=1}^{T} \frac{dL_t(\boldsymbol{s}_t, \boldsymbol{\theta})}{d\boldsymbol{\theta}}\mathbf{u} \tag{10}$$

$$\frac{dL_t(\boldsymbol{s}_t, \boldsymbol{\theta})}{d\boldsymbol{\theta}}\mathbf{u} = \frac{\partial L_t(\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\mathbf{u} + \frac{\partial L_t(\boldsymbol{s}_t, \boldsymbol{\theta})}{\partial \boldsymbol{s}_t}\underbrace{\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}}\mathbf{u}}_{\mathbf{c}_t} \tag{11}$$

$$\mathbf{c}_t = \frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}}\mathbf{u} = \frac{df(\boldsymbol{s}_{t-1}, \boldsymbol{\theta})}{d\boldsymbol{\theta}}\mathbf{u} = \frac{\partial f(\boldsymbol{s}_{t-1}, \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\mathbf{u} + \frac{\partial f(\boldsymbol{s}_{t-1}, \boldsymbol{\theta})}{\partial \boldsymbol{s}_{t-1}}\underbrace{\frac{d\boldsymbol{s}_{t-1}}{d\boldsymbol{\theta}}\mathbf{u}}_{\mathbf{c}_{t-1}} \tag{12}$$



(a) Using the same sequence of direction vectors for ES-Single and DODGE. With a small perturbation scale, ES-Single becomes nearly identical to DODGE, while with a large perturbation scale, it traverses a smoothed landscape.

(b) When using 1000 direction vectors, both DODGE and ES-Single become approximately equivalent to RTRL. Note that here we use a small perturbation scale σ=1e-4 for ES-Single.

Figure 25. Comparing ES-Single and DODGE on the toy 2D regression task from Figure 15. (a) We use the same sequence of random directions for ES-Single and DODGE, where a single direction is sampled at the start of each inner problem and kept fixed over all partial unrolls. Both methods use truncations of length $K = 1$, with total inner problem length $T = 100$. Over the course of an inner problem, DODGE and ES-Single update the outer parameters in a subspace spanned by the direction vector; this leads to the line segments in Fig. (a), each of which shows the progress made during one inner problem. At the end of an inner problem, a new direction is sampled, leading to the piecewise linear structure. As the perturbation scale σ goes to 0, ES-Single becomes nearly identical to DODGE, and is not able to traverse the chaotic regions of the meta-loss landscape. Increasing the perturbation scale allows ES-Single to cross the chaos and reach the optimal meta-objective value. In Fig. (b), we show that increasing the number of direction vectors used DODGE and ES-Single brings both methods very close to exact RTRL. Darker regions represent lower meta-objective values.

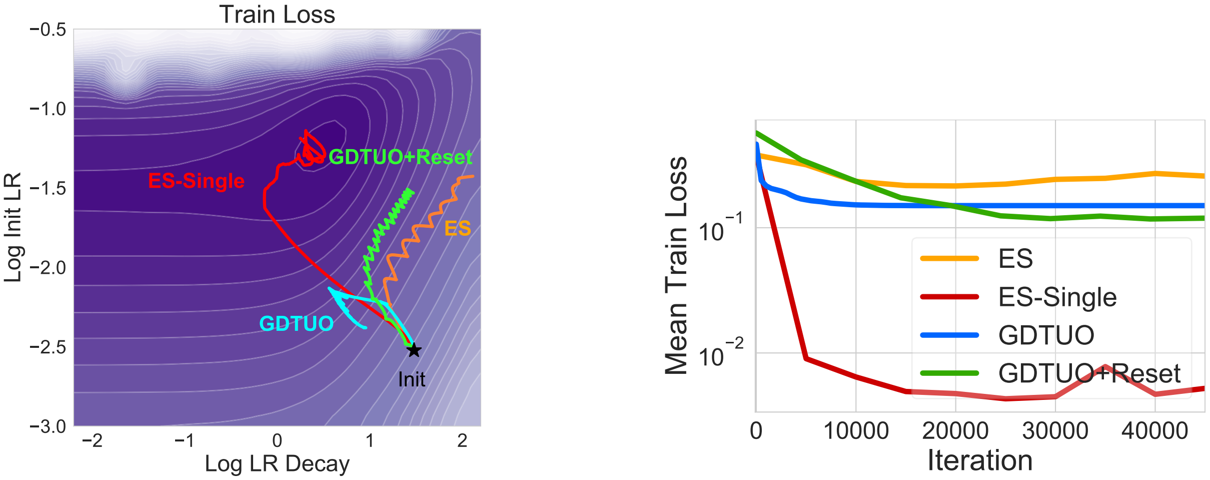
Here, $\mathbf{c}_t$ is a *carry term* that propagates information over the course of a full inner problem. While DODGE and ES-Single quite distinct—ES-Single is gradient-free while DODGE is gradient-based—they share some conceptual similarities. In particular, both perform meta-optimization within a subspace spanned by a set of direction vectors. In DODGE, a few possibilities were given for determining these directions, including drawing samples from an isotropic Gaussian similarly to ES-Single. A critical difference is that ES-Single smooths the outer loss landscape, allowing for optimization over chaotic surfaces that arise in meta-optimization.

As shown in Figure 25, when applying DODGE to a 2D meta-optimization task, it gets stuck when it reaches a chaotic part of the landscape, similarly to the other gradient-based methods (TBPTT, RTRL, and UORO (Tallec & Ollivier, 2017a)).

C.7. Comparison to Hypergradient Descent.

Some algorithms have been proposed to tune optimization hyperparameters (such as learning rates) online during a single training run (e.g., one inner problem as opposed to several), in particular hypergradient descent (HD) (Baydin et al., 2017) and "Gradient Descent: The Ultimate Optimizer" (GDTUO) (Chandra et al., 2022). Both of these adapt optimization hyperparameters based on a 1-step lookahead meta-objective. However, as shown in (Wu et al., 2018), backpropagation through a 1-step unroll may suffer from truncation bias. HD and GDTUO are conceptually similar to our vanilla truncated ES baseline, as they aim to minimize the loss after taking K gradient steps (with $K = 1$); thus, they can be interpreted as gradient-based analogues of truncated ES. In contrast, ES-Single and PES yield unbiased gradient estimates that do not suffer from truncation bias. Here, we used the Github repository of Chandra et al. (2022) (https://github.com/kach/gradient-descent-the-ultimate-optimizer), and applied their method to our task from Section 4.2: tuning the learning rate and decay factor used to train an MLP.

We considered two scenarios for GDTUO: 1) never resetting the inner problem (e.g., the online learning setting they use); and 2) resetting the inner problem every T iterations, while continuing to optimize the outer parameters, which mimics our truncated ES setting. Figures 26a and 26b show that GDTUO behaves similarly to truncated ES, both in terms of the meta-optimization trajectory and the training loss achieved. ES-Single outperforms GDTUO on this task.



(a) Here, we show the trajectories of vanilla truncated ES and ES-Single, alongside the trajectories of two variants of GDTUO. Darker colors denote lower (better) loss values.

(b) Mean training loss values using the same setup as Section 4.2, comparing ES, ES-Single, GDTUO, and GDTUO with inner problem resetting.

Figure 26. Tuning the learning rate and decay factor for SGDm, used to optimize an MLP on MNIST, with the same setup as Section 4.2. We compare the meta-optimization trajectories and training losses that result from using vanilla truncated ES, ES-Single, and two variants of GDTUO (Chandra et al., 2022) that differ with respect to whether the inner problem is reset after T steps.

D. Proofs

D.1. Proof of Unbiasedness

Proposition D.1 (ES-Single is unbiased)**.** *Assume that $L(\boldsymbol{\theta})$ is quadratic and $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ exists. Then, the ES-Single gradient estimator with antithetic sampling is unbiased, that is,* $\text{bias}(\hat{\boldsymbol{g}}^{\textit{ES-Single}}) = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\textit{ES-Single}}\right] - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = 0.$

Proof. By assumption, $L(\boldsymbol{\theta})$ is quadratic, and thus is equivalent to its second-order Taylor series approximation:

$$L(\boldsymbol{\theta} + \boldsymbol{\epsilon}) = L(\boldsymbol{\theta}) + \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) + \frac{1}{2} \boldsymbol{\epsilon}^\top \nabla^2_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \boldsymbol{\epsilon} \tag{13}$$

The antithetic gradient estimator is $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}(L(\boldsymbol{\theta} + \boldsymbol{\epsilon}) - L(\boldsymbol{\theta} - \boldsymbol{\epsilon}))\right]$. We can simplify this expression by noting that:

$$\begin{aligned} \boldsymbol{\epsilon}\left(L(\boldsymbol{\theta} + \boldsymbol{\epsilon}) - L(\boldsymbol{\theta} - \boldsymbol{\epsilon})\right) &= \boldsymbol{\epsilon}\left[L(\boldsymbol{\theta}) + \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) + \frac{1}{2} \boldsymbol{\epsilon}^\top \nabla^2_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \boldsymbol{\epsilon} - L(\boldsymbol{\theta}) + \boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) - \frac{1}{2} \boldsymbol{\epsilon}^\top \nabla^2_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \boldsymbol{\epsilon}\right] \qquad (14) \\ &= \boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \qquad (15) \end{aligned}$$

Thus, the ES-Single gradient is the following Monte Carlo estimate:

$$\hat{\boldsymbol{g}}^{\text{ES-Single}} = \frac{1}{\sigma^2 N} \sum_{i=1}^{N} \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{16}$$

Taking the expectation of this expression, we have:

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Single}}\right] = \frac{1}{\sigma^2 N} \sum_{i=1}^{N} \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top\right]}_{=\sigma^2 \mathbf{I}} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla \boldsymbol{\theta} L(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{17}$$

Thus, $\text{bias}(\hat{\boldsymbol{g}}^{\text{ES-Single}}) = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Single}}\right] - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = 0.$ □

D.2. Variance

Proposition D.2 (ES-Single Variance)**.** *The total variance of ES-Single using antithetic sampling is* $tr(Var(\hat{\boldsymbol{g}}^{\textit{ES-Single}})) = (P+1)\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|^2$*, where* P *is the dimensionality of* $\boldsymbol{\theta}$*.*

Proof. We measure the total variance of the ES-Single estimator, defined as:

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}})) = \text{tr}\left(\mathbb{E}\left[\hat{\boldsymbol{g}}\hat{\boldsymbol{g}}^\top\right] - \mathbb{E}\left[\hat{\boldsymbol{g}}\right]\mathbb{E}\left[\hat{\boldsymbol{g}}\right]^\top\right) = \mathbb{E}\left[\hat{\boldsymbol{g}}^\top\hat{\boldsymbol{g}}\right] - \mathbb{E}\left[\hat{\boldsymbol{g}}\right]^\top\mathbb{E}\left[\hat{\boldsymbol{g}}\right] \tag{18}$$

We assume that the loss L is quadratic, and that we use antithetic samples to estimate the gradient. Here, we consider a single particle pair for simplicity, $N = 1$, such that the estimator can be written as $\hat{\boldsymbol{g}} = \frac{1}{\sigma^2}\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. Because $\hat{\boldsymbol{g}}$ is an unbiased estimator, its expectation is equal to the true gradient, and thus the second term in Eq. 18 is $\mathbb{E}\left[\hat{\boldsymbol{g}}\right]^\top\mathbb{E}\left[\hat{\boldsymbol{g}}\right] = \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|^2$. For the first term, we have:

$$\mathbb{E}\left[\hat{\boldsymbol{g}}^\top\hat{\boldsymbol{g}}\right] = \frac{1}{\sigma^4}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}\boldsymbol{\epsilon}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\right] \tag{19}$$

$$= \frac{1}{\sigma^4}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\right]\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{20}$$

By Isserlis' theorem (see Maheswaranathan et al. (2019)), we have $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\right] = \text{tr}(\Sigma)\Sigma + 2\Sigma^2$, where Σ is the covariance of the perturbation distribution. Because our perturbations are sampled from an isotropic Gaussian, $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, we have $\Sigma = \sigma^2\mathbf{I}$, and thus:

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\right] = \text{tr}(\Sigma)\Sigma + 2\Sigma^2 \tag{21}$$

$$= \text{tr}(\sigma^2\mathbf{I})\sigma^2\mathbf{I} + 2(\sigma^2\mathbf{I})^2 \tag{22}$$

$$= P\sigma^4\mathbf{I} + 2\sigma^4\mathbf{I} \tag{23}$$

$$= (P+2)\sigma^4\mathbf{I} \tag{24}$$

where P is the dimensionality of $\boldsymbol{\theta}$. Plugging this into Eq. 20, we have:

$$\mathbb{E}\left[\hat{\boldsymbol{g}}^\top\hat{\boldsymbol{g}}\right] = \frac{1}{\sigma^4}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\boldsymbol{\epsilon}\boldsymbol{\epsilon}^\top\right]\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{25}$$

$$= \frac{1}{\sigma^4}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top\left[(P+2)\sigma^4\mathbf{I}\right]\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{26}$$

$$= (P+2)\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|^2 \tag{27}$$

Thus, the total variance of the ES-Single estimator is:

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}})) = (P+2)\|\nabla\boldsymbol{\theta} L(\boldsymbol{\theta})\|^2 - \|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|^2 = (P+1)\|\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\|^2 \tag{28}$$

From this, we see that the variance increases linearly in the dimensionality of $\boldsymbol{\theta}$, but does not depend on the number of partial unrolls per inner problem. □

E. Stochastic Computation Graphs

Here, we provide a derivation of ES-Single using the framework of stochastic computation graphs from (Schulman et al., 2015). We also provide a derivation of vanilla truncated ES; the derivation for PES can be found in (Vicol et al., 2021).

ES-Single. For the stochastic computation graph in Figure 4, we have an input node θ that gives rise to a sampled variable $\tilde{\theta}$ which is re-applied over all time steps of the inner problem. Each state $\boldsymbol{s}_t$ depends deterministically on the sampled parameter $\tilde{\theta}$ and the previous state $\boldsymbol{s}_{t-1}$. The losses at each timestep, L_t, are the cost nodes, and the objective is to minimize $L = \sum_{t=1}^{T} L_t$. We leverage Theorem 1 from Schulman et al. (2015), which gives a general expression to compute the gradient of the sum of cost nodes in such a stochastic computation graph:

$$\frac{\partial}{\partial \theta} \mathbb{E}\left[\sum_{c \in \mathcal{C}} c\right] = \mathbb{E}\left[\sum_{w \in \mathcal{S}, \theta \prec^D w} \left(\frac{\partial}{\partial \theta} \log p(w \mid \text{DEPS}_w)\right) \hat{Q}_w + \underbrace{\sum_{c \in \mathcal{C}, \theta \prec^D c} \frac{\partial}{\partial \theta} c(\text{DEPS}_c)}_{=0}\right] \tag{29}$$

Here, $\mathcal{C}$ is the set of cost nodes (the L_t); $\mathcal{S}$ is the set of stochastic nodes (in our case, $\mathcal{S} = \{\tilde{\theta}\}$); DEPS_w denotes the set of nodes that w depends on; $a \prec^D b$ denotes a deterministic dependence of node a on node b (this holds if there are no stochastic nodes along the path from a to b); and $\hat{Q}_w$ is the sum of cost nodes downstream of node w.

In our computation graph, θ does not deterministically influence the cost nodes L_t, so the second term inside the expectation is 0. Thus, for Figure 4, we have:

$$\frac{\partial}{\partial \theta} \mathbb{E}\left[\sum_{t=1}^{T} L_t\right] = \mathbb{E}\left[\left(\frac{\partial}{\partial \theta} \log p(\tilde{\theta} \mid \theta)\right) \hat{Q}_{\tilde{\theta}}\right] \tag{30}$$

Because we sample isotropic Gaussian perturbations to θ, we have $p(\tilde{\theta} \mid \theta) = \mathcal{N}(\tilde{\theta}; \theta, \sigma^2 I) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(\frac{-(\tilde{\theta}-\theta)^2}{2\sigma^2}\right)$. The gradient of the log probability is:

$$\frac{\partial}{\partial \theta} \log p(\tilde{\theta} \mid \theta) = \frac{\partial}{\partial \theta}\left(-\frac{1}{2}\log(2\pi) - \log \sigma - \frac{(\tilde{\theta}-\theta)^2}{2\sigma^2}\right) = \frac{1}{\sigma^2}(\tilde{\theta} - \theta) \tag{31}$$

Using the reparameterization trick, we substitute $\tilde{\theta} = \theta + \epsilon$ where $\epsilon \sim \mathcal{N}(0, \sigma^2 I)$, which yields $\frac{\partial}{\partial \theta} \log p(\tilde{\theta} \mid \theta) = \frac{1}{\sigma^2}(\theta + \epsilon - \theta) = \frac{1}{\sigma^2}\epsilon$. The sum of cost nodes downstream of $\tilde{\theta}$ is $\sum_{t=1}^{T} L_t$ because all cost nodes are downstream of $\tilde{\theta}$. Thus, Theorem 1 from Schulman et al. (2015) gives the following unbiased gradient estimator for the ES-Single computation graph, which is equivalent to the full-unroll ES gradient estimate:

$$\frac{\partial}{\partial \theta} \mathbb{E}\left[\sum_{t=1}^{T} L_t\right] = \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon} \sum_{t=1}^{T} L_t\right] \tag{32}$$

Vanilla Truncated ES. Figure 27 shows the stochastic computation graph corresponding to vanilla truncated ES, which shares similar structure to the PES computation graph (Figure 4), but does not include recurrent connections between successive states $\boldsymbol{s}_{t-1}, \boldsymbol{s}_t$. This ignores crucial structure about the problem, e.g., that it consists of a sequence of unrolls rather than a set of independent minimization problems over separate objectives L_t. Once again leveraging Theorem 1 from Schulman et al. (2015), we derive the following gradient estimator for truncated ES:

$$\frac{\partial}{\partial \theta} \mathbb{E}\left[\sum_{t=1}^{T} L_t\right] = \mathbb{E}_{\epsilon}\left[\sum_{t=1}^{T} \underbrace{\left(\frac{\partial}{\partial \theta} \log p(\theta_t \mid \theta)\right)}_{=\frac{1}{\sigma^2}\epsilon_t} \underbrace{\hat{Q}_{\theta_t}}_{L_t}\right] \tag{33}$$

$$= \mathbb{E}_{\epsilon}\left[\sum_{t=1}^{T} \frac{1}{\sigma^2} \epsilon_t L_t\right] = \frac{1}{\sigma^2} \sum_{t=1}^{T} \mathbb{E}_{\epsilon_t}\left[\epsilon_t L_t(\theta + \epsilon_t)\right] \tag{34}$$

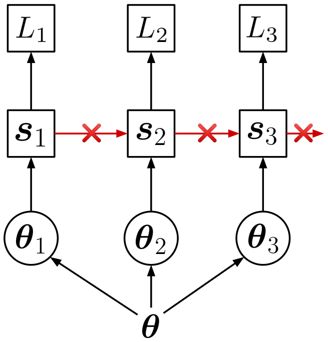


Figure 27. Stochastic computation graph for vanilla truncated ES, which does not include the recurrent connections between states.

F. Generalizing PES and ES-Single by Decoupling Intervals

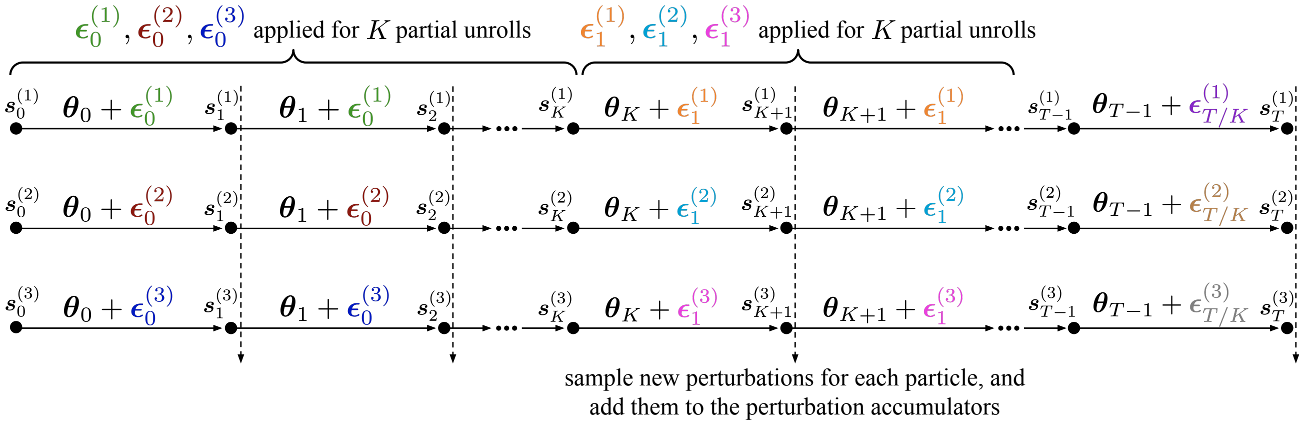

$\epsilon_0^{(1)}, \epsilon_0^{(2)}, \epsilon_0^{(3)}$ applied for K partial unrolls
$\epsilon_1^{(1)}, \epsilon_1^{(2)}, \epsilon_1^{(3)}$ applied for K partial unrolls
$s_0^{(1)}$ $\theta_0 + \epsilon_0^{(1)}$ $s_1^{(1)}$ $\theta_1 + \epsilon_0^{(1)}$ $s_2^{(1)}$ $s_K^{(1)}$ $\theta_K + \epsilon_1^{(1)}$ $s_{K+1}^{(1)}$ $\theta_{K+1} + \epsilon_1^{(1)}$ $s_{T-1}^{(1)}$ $\theta_{T-1} + \epsilon_{T/K}^{(1)}$ $s_T^{(1)}$
$s_0^{(2)}$ $\theta_0 + \epsilon_0^{(2)}$ $s_1^{(2)}$ $\theta_1 + \epsilon_0^{(2)}$ $s_2^{(2)}$ $s_K^{(2)}$ $\theta_K + \epsilon_1^{(2)}$ $s_{K+1}^{(2)}$ $\theta_{K+1} + \epsilon_1^{(2)}$ $s_{T-1}^{(2)}$ $\theta_{T-1} + \epsilon_{T/K}^{(2)}$ $s_T^{(2)}$
$s_0^{(3)}$ $\theta_0 + \epsilon_0^{(3)}$ $s_1^{(3)}$ $\theta_1 + \epsilon_0^{(3)}$ $s_2^{(3)}$ $s_K^{(3)}$ $\theta_K + \epsilon_1^{(3)}$ $s_{K+1}^{(3)}$ $\theta_{K+1} + \epsilon_1^{(3)}$ $s_{T-1}^{(3)}$ $\theta_{T-1} + \epsilon_{T/K}^{(3)}$ $s_T^{(3)}$
sample new perturbations for each particle, and
add them to the perturbation accumulators

Figure 28. Computation graph for a generalization of ES-Single and PES, in which the interval at which we update the outer parameters is decoupled from the interval at which we sample new perturbations and update the perturbation accumulator for each particle. In particular, vanilla PES in each partial unroll samples a new perturbation, updates the accumulator, and updates the outer parameters; in contrast, here we update the outer parameters multiple times using the same perturbation, and only update the perturbations (and the accumulators) every K outer parameter updates.

Here, we propose a generalization of both ES-Single and PES, that decouples the interval at which we update the perturbation accumulator from the interval at which we update the outer parameters. Recall that ES-Single has constant variance regardless of the number of partial unrolls per inner problem (Figure 1). For long-horizon inner problems optimized using short truncations—yielding a large number of partial unrolls per problem—ES-Single can have substantially lower variance than PES. However, for certain scenarios (including the the real sequence from the PTB dataset) and unroll lengths (e.g. such that we have ~ 10 unrolls per problem), PES has lower variance than ES-Single (Figure 1). This motivated us to consider an algorithm that generalizes both PES and ES-Single. In particular, we introduce another hyperparameter, Ω, that specifies the meta-update interval, while K denotes the interval at which new perturbations are sampled and at which the perturbation accumulator is updated. Many algorithms of interest can be obtained as special cases, by setting K and Ω appropriately. Let T be the length of a full inner problem. Then, 1) if $K = \Omega = T$, we recover full-unroll ES; 2) if $K = \Omega$ and $K < T$, we recover PES; 3) if $K = T$ and $\Omega < T$, we recover ES-Single; and 4) if $K, \Omega < T$ and $\Omega < K$, we obtain a new estimator with a combination of the properties of ES-Single and PES. Algorithm 29 formally describes the latter case. Similarly to PES and ES-Single, separate states $\boldsymbol{s}^{(i)}$ are maintained for each particle over the course of an inner problem. Like PES, a perturbation accumulator $\boldsymbol{\xi}^{(i)}$ is maintained for each particle, and However, rather than sampling perturbations $\boldsymbol{\epsilon}^{(i)}$ per particle for each partial unroll, new perturbations are only sampled every M partial unrolls. That is, the same perturbation is re-applied for M consecutive unrolls. Correspondingly, the perturbation accumulator is only updated once every M unrolls.

General Stochastic Computation Graph. In Figure 30, we provide the stochastic computation graph for the generalized estimator, that re-uses the same outer parameter perturbations for a sequence of K partial unrolls before re-sampling the

Algorithm 3 Truncated Evolution Strategies (ES) applied to partial unrolls of a computation graph.

```
Input: $\boldsymbol{s}_0$, initial state
       $K$, truncation length for partial unrolls
       $N$, number of particles
       $\sigma$, standard deviation of perturbations
       $\alpha$, learning rate for outer optimization
Initialize $\boldsymbol{s} = \boldsymbol{s}_0$
while inner problem not finished do
    $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \mathbf{0}$
    for $i = 1, \ldots, N$ do
        $\boldsymbol{\epsilon}^{(i)} = \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 I) & i \text{ odd} \\ -\boldsymbol{\epsilon}^{(i-1)} & i \text{ even} \end{cases}$
        $\hat{L}_K^{(i)} \leftarrow \text{unroll}(\boldsymbol{s}, \boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}, K)$
        $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \hat{\boldsymbol{g}}^{\text{ES}} + \boldsymbol{\epsilon}^{(i)} \hat{L}_K^{(i)}$
    end for
    $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{ES}}$
    $\boldsymbol{s} \leftarrow \text{unroll}(\boldsymbol{s}, \boldsymbol{\theta}, K)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{ES}}$
end while
```

Algorithm 4 Generalization of ES-Single and PES, with an arbitrary re-sampling interval M.

```
Input: $\boldsymbol{s}_0$, initial state
       $K$, truncation length for partial unrolls
       $M$, re-sampling interval
       $N$, number of particles
       $\sigma$, standard deviation of perturbations
       $\alpha$, learning rate for outer optimization
Initialize $\boldsymbol{s}^{(i)} = \boldsymbol{s}_0$ for $i \in \{1, \ldots, N\}$
Initialize $\boldsymbol{\xi}^{(i)} \leftarrow \mathbf{0}$ for $i \in \{1, \ldots, N\}$
while inner problem not finished, iteration $j$ do
    if $j \mod M = 0$ then
        for $i = 1, \ldots, N$ do
            $\boldsymbol{\epsilon}^{(i)} = \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 I) & i \text{ odd} \\ -\boldsymbol{\epsilon}^{(i-1)} & i \text{ even} \end{cases}$
            $\boldsymbol{\xi}^{(i)} \leftarrow \boldsymbol{\xi}^{(i)} + \boldsymbol{\epsilon}^{(i)}$
        end for
    end if
    $\hat{\boldsymbol{g}}^{\text{ES-Gen}} \leftarrow \mathbf{0}$
    for $i = 1, \ldots, N$ do
        $\boldsymbol{s}^{(i)}, \hat{L}_K^{(i)} \leftarrow \text{unroll}(\boldsymbol{s}^{(i)}, \boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}, K)$
        $\hat{\boldsymbol{g}}^{\text{ES-Gen}} \leftarrow \hat{\boldsymbol{g}}^{\text{ES-Gen}} + \boldsymbol{\xi}^{(i)} \hat{L}_K^{(i)}$
    end for
    $\hat{\boldsymbol{g}}^{\text{ES-Gen}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{ES-Gen}}$

    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{ES-Gen}}$
end while
```

Figure 29. **A comparison of vanilla ES and the generalized form of PES and ES-Single**, applied to partial unrolls of a computation graph. The conditional statement for $\boldsymbol{\epsilon}^{(i)}$ is used to implement antithetic sampling. Differences between the two algorithms are highlighted in red. While ES samples different perturbations for each particle in each partial unroll, ES-Single re-applies the same perturbation over a sequence of partial unrolls, and every M unrolls, it updates the perturbation accumulator and re-samples the perturbations.

outer parameters. The resulting unbiased gradient estimator is:

$$\frac{\partial}{\partial \boldsymbol{\theta}} \mathbb{E}\left[\sum_{t=1}^{T} L_t\right] = \mathbb{E}\left[\sum_{t=1}^{T/K} \left(\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t \mid \boldsymbol{\theta})\right) \hat{Q}_{\boldsymbol{\theta}_t}\right] \tag{35}$$

$$= \frac{1}{\sigma^2} \mathbb{E}\left[\sum_{t=1}^{T/K} \boldsymbol{\epsilon}_t \left(\sum_{\tau=kt+1}^{T} L_\tau\right)\right] \tag{36}$$

$$= \frac{1}{\sigma^2} \mathbb{E}\left[\boldsymbol{\epsilon}_1 (L_1 + L_2 + \cdots + L_K) + (\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_2)(L_{K+1} + \cdots + L_{2K}) + \cdots + (\boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_{T/K}) L_T\right] \tag{37}$$

$$= \frac{1}{\sigma^2} \mathbb{E}\left[\sum_{t=1}^{T/K} \left(\sum_{\tau=1}^{t} \boldsymbol{\epsilon}_\tau\right) (L_{tK+1} + \cdots + L_{2tK})\right] \tag{38}$$

This estimator has variance equivalent to PES where the number of unrolls per inner problem is K. The main benefit is that it allows for more frequent updates to the outer parameters, while maintaining this fixed variance, which is determined by a hyperparameter that can be tuned independently of the update frequency.

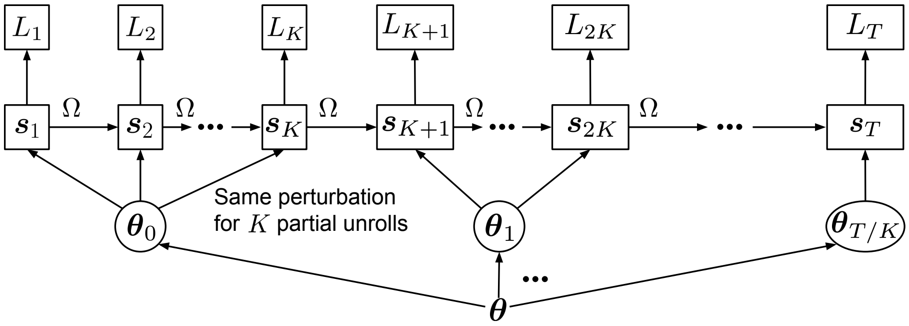


Figure 30. Computation graph corresponding to a generalization of ES-Single and PES, where we re-apply the same perturbed outer parameters (and use the same accumulated perturbations) over a sequence of K partial unrolls of the inner problem. After each sequence of unrolls, the perturbations are re-sampled and the perturbation accumulator is updated. Note that the computation graphs for ES-Single and PES shown in Figure 4 are special cases corresponding to two extremes: 1) in ES-Single, the same perturbation is applied across all partial unrolls, yielding a single stochastic node $\tilde{\boldsymbol{\theta}}$ for the inner problem—in this case, the perturbation accumulators are equivalent to the perturbations themselves (e.g., they are the sum of only one term); and 2) in PES, a different perturbation is applied in each partial unroll—yielding T stochastic nodes $\{\boldsymbol{\theta}_t\}_{t=1}^T$ per inner problem—and the accumulators are updated after each unroll.

G. Variance Analysis of a Generalized Estimator

In this section, we introduce an estimator that generalizes ES-Single and PES by combining a single perturbation that is kept fixed across all partial unrolls (as in ES-Single) with perturbations that are sampled independently for each partial unroll (as in PES). One way to obtain this estimator is to consider the stochastic computation graph in Figure 31. Here, each stochastic node $\boldsymbol{\theta}_t$ and $\tilde{\boldsymbol{\theta}}$ depends only on $\boldsymbol{\theta}$, e.g., $\text{DEPS}_{\boldsymbol{\theta}_t} = \{\boldsymbol{\theta}\} \forall t$ and $\text{DEPS}_{\tilde{\boldsymbol{\theta}}} = \{\boldsymbol{\theta}\}$. Then, the gradient estimator is:

$$\boldsymbol{g}^{\text{ES-Gen}} = \mathbb{E}\left[\sum_{w \in \mathcal{S}, \boldsymbol{\theta} \prec^D w} \left(\frac{\partial}{\partial \boldsymbol{\theta}} \log p(w \mid \text{DEPS}_w)\right) \hat{Q}_w\right] \tag{39}$$

$$= \mathbb{E}\left[\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\tilde{\boldsymbol{\theta}} \mid \boldsymbol{\theta}) \hat{Q}_{\tilde{\boldsymbol{\theta}}} + \sum_{t=1}^{T} \left(\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\boldsymbol{\theta}_t \mid \boldsymbol{\theta})\right) \hat{Q}_{\boldsymbol{\theta}_t}\right] \tag{40}$$

$$= \mathbb{E}\left[\frac{1}{\sigma^2} \boldsymbol{\epsilon}_s \left(\sum_{t=1}^{T} L_t\right) + \sum_{t=1}^{T} \frac{1}{\sigma^2} \boldsymbol{\epsilon}_t \left(\sum_{\tau=t}^{T} L_\tau\right)\right] \tag{41}$$

$$= \frac{1}{\sigma^2} \mathbb{E}\left[\sum_{t=1}^{T} \left(\boldsymbol{\epsilon}_s + \sum_{\tau=1}^{t} \boldsymbol{\epsilon}_\tau\right) L_t\right] \tag{42}$$

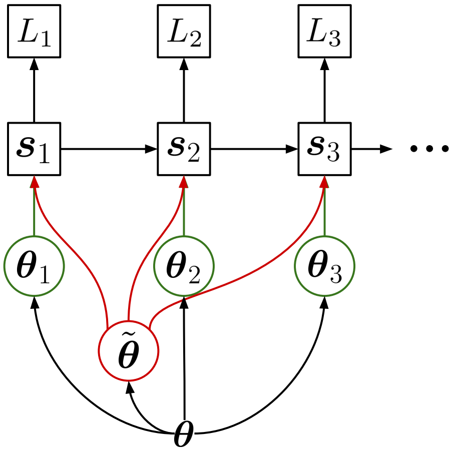


Figure 31. Stochastic computation graph for a generalization of PES and ES-Single, where we consider two types of perturbations: 1) a single perturbation that is applied in each unroll over the course of a full inner problem; and 2) a separate perturbation sampled for each partial unroll.

where $\boldsymbol{\epsilon}_s$ denotes a single perturbation that is sampled at the beginning of an inner problem and is kept fixed over the course of all partial unrolls, and $\boldsymbol{\epsilon}_t$ denotes an independent perturbation sampled in a particular partial unroll, at step t. We will analyze a variant of this estimator that weights the contributions of these perturbations using hyperparameters α and β, allowing one to interpolate between PES and ES-Single.

In the following, we adopt notation from Vicol et al. (2021): rather than writing the loss at step t as a function of the parameters $\boldsymbol{\theta}$ and state $\boldsymbol{s}_t$, $L_t(\boldsymbol{s}_t, \boldsymbol{\theta})$, we drop the dependence on $\boldsymbol{s}_t$ and explicitly denote the dependence of L_t on the sequence of applications of $\boldsymbol{\theta}$ over time, $L_t(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \dots, \boldsymbol{\theta}_t)$. This allows us to keep track of how the applications of $\boldsymbol{\theta}$ contribute to the total loss gradient. In addition, we denote by Θ a matrix whose rows are the per-timestep parameters $\boldsymbol{\theta}_t$, where $\boldsymbol{\theta}_t = \boldsymbol{\theta}, \forall t$. Then, we can write $L_t(\Theta)$ as shorthand for $L_t(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t)$. Finally, we use $\boldsymbol{\xi}_t$ to denote the PES perturbation accumulator, that sums the perturbations up to step t, $\boldsymbol{\xi}_t = \sum_{\tau=1}^{t} \boldsymbol{\epsilon}_\tau$. Assuming that the objective is quadratic, and that we use antithetic sampling, we can write the estimator as:

$$\boldsymbol{g}^{\text{ES-Gen}} = \frac{1}{\alpha^2\sigma^2 + \beta^2\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\left(\alpha\boldsymbol{\epsilon}_s + \beta\sum_{\tau=1}^{t}\boldsymbol{\epsilon}_t\right)\text{vec}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_{1..t})^\top\nabla_{\text{vec}(\Theta_{1..t})}L_t(\Theta)\right] \tag{43}$$

where vec denotes the vectorization operator. The expression $\text{vec}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_{1..t})$ uses broadcasting to add the single perturbation $\boldsymbol{\epsilon}_s$ to each of the t per-unroll perturbations in $\boldsymbol{\epsilon}_{1..t}$. The hyperparameters α and β weight the contributions of the perturbations corresponding to ES-Single and PES. This estimator generalizes both approaches: setting $\alpha = 1, \beta = 0$ recovers ES-Single, while setting $\alpha = 0, \beta = 1$ recovers PES.

Unbiasedness. Let $\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ denote the Monte Carlo estimate of $\boldsymbol{g}^{\text{ES-Gen}}$ using N particles: $\hat{\boldsymbol{g}}^{\text{ES-Gen}} = \frac{1}{N(\alpha^2\sigma^2+\beta^2\sigma^2)}\sum_{i=1}^{N}\sum_{t=1}^{T}(\alpha\boldsymbol{\epsilon}_s^{(i)} + \beta\boldsymbol{\xi}_t^{(i)})\sum_{\tau=1}^{t}(\alpha\boldsymbol{\epsilon}_s^{(i)} + \beta\boldsymbol{\epsilon}_\tau^{(i)})^\top\nabla_{\boldsymbol{\theta}_\tau}L_t(\Theta)$. Here, we prove that $\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ is unbiased under the same assumptions used to show the unbiasedness of PES and ES-Single.

Proposition G.1 ($\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ is unbiased)**.** *Assume that the loss $L(\boldsymbol{\theta})$ is quadratic and $\nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta})$ exists. Then, the ES-Gen gradient estimator with antithetic sampling is unbiased, that is, bias$(\hat{\boldsymbol{g}}^{\text{ES-Gen}}) = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right] - \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) = 0$, where the expectation is taken with respect to both the single perturbation $\boldsymbol{\epsilon}_s$ and the per-unroll perturbations $\boldsymbol{\epsilon}_t$.*

Proof. First, we expand out the term $\text{vec}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_{1..t})^\top\nabla_{\text{vec}(\Theta_{1..t}}L_t(\Theta)$ as follows:

$$\text{vec}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_{1..t})^\top\nabla_{\text{vec}(\Theta_{1..t}}L_t(\Theta) = \sum_{\tau=1}^{t}\left(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_\tau\right)^\top\nabla_{\boldsymbol{\theta}_\tau}L_t(\Theta) \tag{44}$$

Plugging this into the expression for $\boldsymbol{g}^{\text{ES-Gen}}$, we have:

$$\boldsymbol{g}^{\text{ES-Gen}} = \frac{1}{\alpha^2\sigma^2 + \beta^2\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\left(\alpha\boldsymbol{\epsilon}_s + \beta\underbrace{\left(\sum_{\tau=1}^{t}\boldsymbol{\epsilon}_\tau\right)}_{\boldsymbol{\xi}_t}\right)\left(\sum_{\tau=1}^{t}(\alpha\boldsymbol{\epsilon}_s + \boldsymbol{\epsilon}_\tau)^\top\nabla_{\boldsymbol{\theta}_\tau}L_t(\Theta)\right)\right] \tag{45}$$

$$= \frac{1}{\alpha^2\sigma^2 + \beta^2\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_t)\sum_{\tau=1}^{t}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_\tau)^\top\nabla_{\boldsymbol{\theta}_\tau}L_t(\Theta)\right] \tag{46}$$

Expanding out $\sum_{\tau=1}^{t}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_\tau)^\top\nabla_{\boldsymbol{\theta}_\tau}L_t(\Theta)$, we have:

$$\sum_{\tau=1}^{t}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_\tau)^\top\nabla_{\boldsymbol{\theta}_\tau}L_t(\Theta) = \alpha\boldsymbol{\epsilon}_s^\top\nabla_{\boldsymbol{\theta}_1}L_t + \cdots + \alpha\boldsymbol{\epsilon}_s\nabla_{\boldsymbol{\theta}_t}L_t + \beta\boldsymbol{\epsilon}_1^\top\nabla_{\boldsymbol{\theta}_1}L_t + \cdots + \beta\boldsymbol{\epsilon}_t\nabla_{\boldsymbol{\theta}_t}L_t \tag{47}$$

Next, multiplying these terms by $\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_t$, we have:

$$(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_t)\left(\sum_{\tau=1}^{t}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_\tau)^\top\nabla_{\boldsymbol{\theta}_\tau}L_t(\Theta)\right) = \underbrace{\alpha^2 + \boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top\nabla_{\boldsymbol{\theta}_1}L_t + \alpha^2\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top\nabla_{\boldsymbol{\theta}_2}L_t + \cdots + \alpha\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top\nabla_{\boldsymbol{\theta}_t}L_t}_{①} \tag{48}$$

$$+ \underbrace{\alpha\beta\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_1^\top\nabla_{\boldsymbol{\theta}_1}L_t + \alpha\beta\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_2^\top\nabla_{\boldsymbol{\theta}_2}L_t + \cdots + \alpha\beta\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_t^\top\nabla_{\boldsymbol{\theta}_t}L_t}_{②} \tag{49}$$

$$+ \underbrace{\alpha\beta\boldsymbol{\xi}_t\boldsymbol{\epsilon}_s^\top\nabla_{\boldsymbol{\theta}_1}L_t + \alpha\beta\boldsymbol{\xi}_t\boldsymbol{\epsilon}_s^\top\nabla_{\boldsymbol{\theta}_2}L_t + \cdots + \alpha\beta\boldsymbol{\xi}_t\boldsymbol{\epsilon}_s^\top\nabla_{\boldsymbol{\theta}_t}L_t}_{③} \tag{50}$$

$$+ \underbrace{\beta^2\boldsymbol{\xi}_t\boldsymbol{\epsilon}_1^\top\nabla_{\boldsymbol{\theta}_1}L_t + \beta^2\boldsymbol{\xi}_t\boldsymbol{\epsilon}_2^\top\nabla_{\boldsymbol{\theta}_2}L_t + \cdots + \beta^2\boldsymbol{\xi}_t\boldsymbol{\epsilon}_t^\top\nabla_{\boldsymbol{\theta}_t}L_t}_{④} \tag{51}$$

The expectations of these four terms are as follows:

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[①\right] = \alpha^2 \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_s \boldsymbol{\epsilon}_s^\top\right]}_{=\sigma^2 \mathbf{I}} \nabla_{\boldsymbol{\theta}_1} L_t + \cdots + \alpha^2 \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_s \boldsymbol{\epsilon}_s^\top\right]}_{=\sigma^2 \mathbf{I}} \nabla_{\boldsymbol{\theta}_t} L_t \tag{52}$$

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[②\right] = \alpha\beta \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_s \boldsymbol{\epsilon}_1^\top\right]}_{=0} \nabla_{\boldsymbol{\theta}_1} L_t + \cdots + \alpha\beta \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_s \boldsymbol{\epsilon}_t^\top\right]}_{=0} \nabla_{\boldsymbol{\theta}_t} L_t = 0 \tag{53}$$

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[③\right] = \alpha\beta \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_s^\top\right]}_{=0} \nabla_{\boldsymbol{\theta}_1} L_t + \cdots + \alpha\beta \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_s^\top\right]}_{=0} \nabla_{\boldsymbol{\theta}_t} L_t \tag{54}$$

$$\mathbb{E}_{\boldsymbol{\epsilon}}\left[④\right] = \beta^2 \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_1^\top\right]}_{=\sigma^2 \mathbf{I}} \nabla_{\boldsymbol{\theta}_1} L_t + \cdots + \beta^2 \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_t^\top\right]}_{=\sigma^2 \mathbf{I}} \nabla_{\boldsymbol{\theta}_t} L_t \tag{55}$$

Note that $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_s^\top\right] = 0$ because $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_s^\top\right] = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau=1}^t \boldsymbol{\epsilon}_\tau\right) \boldsymbol{\epsilon}_s^\top\right] = \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_s^\top\right]}_{=0} + \cdots + \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_s^\top\right]}_{=0} = 0$ Similarly, $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_1^\top\right] = 0$ because $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\xi}_t \boldsymbol{\epsilon}_1^\top\right] = \mathbb{E}_{\boldsymbol{\epsilon}}\left[(\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_t)\boldsymbol{\epsilon}_1^\top\right] = \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_1^\top\right]}_{=\sigma^2 \mathbf{I}} + \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_2 \boldsymbol{\epsilon}_1^\top\right]}_{=0} + \cdots + \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_1^\top\right]}_{=0}$.

Combining the four expectations, we have:

$$\alpha^2\sigma^2 \nabla_{\boldsymbol{\theta}_1} L_t + \cdots + \alpha^2\sigma^2 \nabla_{\boldsymbol{\theta}_t} L_t + \beta^2\sigma^2 \nabla_{\boldsymbol{\theta}_1} L_t + \cdots + \beta^2\sigma^2 \nabla_{\boldsymbol{\theta}_t} L_t = (\alpha^2\sigma^2 + \beta^2\sigma^2) \sum_{\tau=1}^t \nabla_{\boldsymbol{\theta}_\tau} L_t(\Theta) \tag{56}$$

The expectation of $\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ is:

$$\begin{aligned}
\mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right] &= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\frac{1}{N(\alpha^2\sigma^2 + \beta^2\sigma^2)} \sum_{i=1}^N \sum_{t=1}^T (\alpha\boldsymbol{\epsilon}_s^{(i)} + \beta\boldsymbol{\xi}_t^{(i)}) \sum_{\tau=1}^t (\alpha\boldsymbol{\epsilon}_s^{(i)} + \beta\boldsymbol{\epsilon}_\tau^{(i)})^\top \nabla_{\boldsymbol{\theta}_\tau} L_t(\Theta)\right] && (57)\\
&= \frac{1}{N(\alpha^2\sigma^2 + \beta^2\sigma^2)} \sum_{i=1}^N \sum_{t=1}^T \mathbb{E}_{\boldsymbol{\epsilon}}\left[(\alpha\boldsymbol{\epsilon}_s^{(i)} + \beta\boldsymbol{\xi}_t^{(i)}) \sum_{\tau=1}^t (\alpha\boldsymbol{\epsilon}_s^{(i)} + \beta\boldsymbol{\epsilon}_\tau^{(i)})^\top \nabla_{\boldsymbol{\theta}_\tau} L_t(\Theta)\right] && (58)\\
&= \frac{1}{N(\alpha^2\sigma^2 + \beta^2\sigma^2)} \sum_{i=1}^N \sum_{t=1}^T (\alpha^2\sigma^2 + \beta^2\sigma^2) \sum_{\tau=1}^t \nabla_{\boldsymbol{\theta}_\tau} L_t(\Theta) && (59)\\
&= \frac{1}{N} \sum_{i=1}^N \left(\sum_{t=1}^T \sum_{\tau=1}^t \nabla_{\boldsymbol{\theta}_\tau} L_t(\Theta)\right) = \sum_{t=1}^T \sum_{\tau=1}^t \nabla_{\boldsymbol{\theta}_\tau} L_t(\Theta) && (60)\\
&= \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) && (61)
\end{aligned}$$

Thus, $\text{bias}(\hat{\boldsymbol{g}}^{\text{ES-Gen}}) = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right] - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = 0$, so $\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ is unbiased. □

G.1. Variance

We assume that the inner problem is quadratic, and that we use antithetic sampling. Given a single particle pair for antithetic sampling, we have the following estimator:

$$\hat{\boldsymbol{g}}^{\text{ES-Gen}} = \frac{1}{\alpha^2\sigma^2 + \beta^2\sigma^2} \sum_{t=1}^T (\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_t) \text{vec}(\alpha\boldsymbol{\epsilon} + \beta\boldsymbol{\epsilon}_{1..t})^\top \nabla_{\text{vec}(\Theta_{1..t})} L_t(\Theta) \tag{62}$$

Similarly to Maheswaranathan et al. (2019) and Vicol et al. (2021), we quantify the variance of $\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ using the total variance $\text{tr}(\text{Var}(\hat{\boldsymbol{g}}^{\text{ES-Gen}}))$:

$$\begin{aligned}
\text{tr}(\text{Var}(\hat{\boldsymbol{g}}^{\text{ES-Gen}})) &= \text{tr}\left(\mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}} \hat{\boldsymbol{g}}^{\text{ES-Gen}\top}\right] - \mathbb{E}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right] \mathbb{E}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right]^\top\right) && (63)\\
&= \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}\top} \hat{\boldsymbol{g}}^{\text{ES-Gen}}\right]}_{①} - \underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right]^\top \mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right]}_{②} && (64)
\end{aligned}$$

Term ② is simple, because $\hat{\boldsymbol{g}}^{\text{ES-Gen}}$is unbiased, so $\mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right] = \nabla_{\boldsymbol{\theta}} L(\Theta)$. Thus,

$$② = \mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right]^\top \mathbb{E}_{\boldsymbol{\epsilon}}\left[\hat{\boldsymbol{g}}^{\text{ES-Gen}}\right] = \nabla_{\boldsymbol{\theta}} L(\Theta)^\top \nabla_{\boldsymbol{\theta}} L(\Theta) = \|\nabla_{\boldsymbol{\theta}} L(\Theta)\|^2 \tag{65}$$

To deal with term ①, we will decompose $\hat{\boldsymbol{g}}^{\text{ES-Gen}\top}\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ into simpler expressions and use the linearity of expectation to compute each component. We will use the shorthand $\boldsymbol{v}_t \equiv \text{vec}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_{1..t})$ and $\boldsymbol{g}_t \equiv \nabla_{\text{vec}(\Theta_{1..t})} L_t(\Theta)$. Note that $\boldsymbol{v}_t^\top \boldsymbol{g}_t = \sum_{\tau=1}^{t}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_\tau)^\top \nabla_{\boldsymbol{\epsilon}_\tau} L_t(\Theta)$. Next, we expand out $\hat{\boldsymbol{g}}^{\text{ES-Gen}\top}\hat{\boldsymbol{g}}^{\text{ES-Gen}}$ using this shorthand:

$$\hat{\boldsymbol{g}}^{\text{ES-Gen}\top}\hat{\boldsymbol{g}}^{\text{ES-Gen}} = \frac{1}{(\alpha^2\sigma^2 + \beta^2\sigma^2)^2}\left(\sum_{t=1}^{T}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_t)\boldsymbol{v}_t^\top \boldsymbol{g}_t\right)^\top \left(\sum_{t=1}^{T}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_t)\boldsymbol{v}_t^\top \boldsymbol{g}_t\right) \tag{66}$$

$$= \frac{1}{(\alpha^2\sigma^2 + \beta^2\sigma^2)^2}\left[\underbrace{\boldsymbol{g}_1^\top \boldsymbol{v}_1(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_1)^\top(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_1)\boldsymbol{v}^\top \boldsymbol{g}_1}_{\textcircled{a}} + \underbrace{\boldsymbol{g}_1^\top \boldsymbol{v}_1(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_1)^\top(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_2)\boldsymbol{v}_2^\top \boldsymbol{g}_2}_{\textcircled{b}} + \cdots\right] \tag{67}$$

There are two types of terms in this expression: terms of type ⓐ, that have the form $\boldsymbol{g}_i^\top \boldsymbol{v}_i(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_i)^\top(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_i)\boldsymbol{v}_i^\top \boldsymbol{g}_i$, and terms of type ⓑ that have the form $\boldsymbol{g}_i^\top \boldsymbol{v}_i(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_i)^\top(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_j)\boldsymbol{v}_j^\top \boldsymbol{g}_j$ where $i \neq j$.

G.1.1. Terms of type ⓐ.

First, note that $(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_i)^\top(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_i)$ can be expanded as follows:

$$(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_i)^\top(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\xi}_i) = \alpha^2\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s + 2\alpha\beta\boldsymbol{\epsilon}_s^\top(\boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_i) + \beta^2\left(\sum_{m=1}^{i}\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_m + \sum_{m\le i, n\le i, m\neq n}\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_n\right) \tag{68}$$

To simplify notation, we will use the shorthand $W = \sum_{m=1}^{i}(\alpha\boldsymbol{\epsilon}_s + \beta\boldsymbol{\epsilon}_m)^\top \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)$. Then, for terms of type ⓐ, we need to compute:

$$\underbrace{W^\top(\alpha^2\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s)W}_{\textcircled{i}} + \underbrace{W^\top(2\alpha\beta\boldsymbol{\epsilon}_s^\top(\boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_i))W}_{\textcircled{ii}} + \underbrace{\beta^2 W^\top\left(\sum_{m=1}^{i}\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_m\right)W}_{\textcircled{iii}} + \underbrace{\beta^2 W^\top\left(\sum_{m\le i, n\le i, m\neq n}\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_n\right)W}_{\textcircled{iv}} \tag{69}$$

Term ⓘ. There are two types of terms that have non-zero expectation: ones with the structure $\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top$, and ones with the structure $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_m^\top$. The contribution from the first type of term is:

$$\alpha^4\sigma^4(P+2)\left(\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m} L_i(\Theta)\right)^\top\left(\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m} L_i(\Theta)\right) \tag{70}$$

And the contribution from the second type of term is:

$$\alpha^2\beta^2\sigma^4 P\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top\nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \tag{71}$$

Term ⓘⓘ. Here, we have two types of terms that have non-zero expectations: ones with the structure $\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_m^\top$ and ones with the structure $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_s^\top$. We will have i terms of each of these sub-types. The total contribution of these terms is:

$$4\alpha^2\beta^2\sigma^4\sum_{m=1}^{i}\left(\sum_{m'=1}^{i}\nabla_{\boldsymbol{\theta}_{m'}} L_i(\Theta)\right)^\top\nabla_{\boldsymbol{\theta}_m} L_i(\Theta) = 4\alpha^2\beta^2\sigma^4\left(\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m} L_i(\Theta)\right)^\top\left(\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m} L_i(\Theta)\right) \tag{72}$$

Term $\textcircled{iii}$. Here, we will have non-zero terms of three types: $\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_s^\top$, $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_m^\top$, and $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_n^\top\boldsymbol{\epsilon}_n\boldsymbol{\epsilon}_m^\top$. The contribution from the first type of term is:

$$\alpha^2\beta^2\sigma^4 Pi\left(\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)\right)^\top\left(\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)\right) \tag{73}$$

The contribution from the second type of term is:

$$\beta^4\sigma^4(P+2)\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)^\top\nabla_{\boldsymbol{\theta}_m}L_i(\Theta) \tag{74}$$

And the contribution from the third type of term ($\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_n^\top\boldsymbol{\epsilon}_n\boldsymbol{\epsilon}_m$) is:

$$\beta^4\sigma^4 P\sum_{n=1}^{i}\left(\sum_{m\leq i, m\neq n}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)^\top\nabla_{\boldsymbol{\theta}_n}L_i(\Theta)\right) \tag{75}$$

Term $\textcircled{iv}$. Here, the nonzero terms arise from two structures, $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_n\boldsymbol{\epsilon}_n^\top$ and $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_n^\top\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_n^\top$. The combined contribution from both types of terms is:

$$2\beta^4\sigma^4\sum_{m\leq i, n\leq i, m\neq n}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)^\top\nabla_{\boldsymbol{\theta}_n}L_i(\Theta) \tag{76}$$

G.1.2. Terms of Type $\textcircled{b}$.

Next, we are interested in terms of type $\textcircled{b}$, which have the form $\boldsymbol{g}_i^\top\boldsymbol{v}_i(\alpha\boldsymbol{\epsilon}_s+\beta\boldsymbol{\xi}_i)^\top(\alpha\boldsymbol{\epsilon}_s+\beta\boldsymbol{\xi}_j)\boldsymbol{v}_j^\top\boldsymbol{g}_j$ where $i\neq j$. In this subsection, we will denote the minimum of i and j by $r\equiv\min(i,j)$. We can expand $(\alpha\boldsymbol{\epsilon}_s+\beta\boldsymbol{\xi}_i)^\top(\alpha\boldsymbol{\epsilon}_s+\boldsymbol{\xi}_j)$ as follows:

$$\begin{aligned}
(\alpha\boldsymbol{\epsilon}_s+\beta\boldsymbol{\xi}_i)^\top(\alpha\boldsymbol{\epsilon}_s+\beta\boldsymbol{\xi}_j) &= (\alpha\boldsymbol{\epsilon}_s+\beta(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_i))^\top(\alpha\boldsymbol{\epsilon}_s+\beta(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_j)) && (77)\\
&= \alpha^2\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s+\alpha\beta\boldsymbol{\epsilon}_s^\top(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_i)+\alpha\beta\boldsymbol{\epsilon}_s^\top(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_j)+ && (78)\\
&\quad+\beta^2(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_i)^\top(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_j) && (79)\\
&= \underbrace{\alpha\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s}_{\textcircled{i}}+\underbrace{\alpha\beta\boldsymbol{\epsilon}_s^\top(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_i)}_{\textcircled{ii}}+\underbrace{\alpha\beta\boldsymbol{\epsilon}_s^\top(\boldsymbol{\epsilon}_1+\cdots+\boldsymbol{\epsilon}_j)}_{\textcircled{iii}} && (80)\\
&\quad+\beta^2\left(\underbrace{\sum_{m=1}^{r}\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_m}_{\textcircled{iv}}+\underbrace{\sum_{m\leq i, n\leq j m\neq n}\boldsymbol{\epsilon}_m^\top\boldsymbol{\epsilon}_n}_{\textcircled{v}}\right) && (81)
\end{aligned}$$

Term $\textcircled{i}$. We have two types of terms with non-zero expectations: $\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top$ and $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_m^\top$. In particular, we have a single instance of the former type of term, and r instances of the latter type. The total contribution of the first term is:

$$\alpha^4\sigma^4(P+2)\left(\sum_{m=1}^{i}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)\right)^\top\left(\sum_{n=1}^{j}\nabla_{\boldsymbol{\theta}_n}L_j(\Theta)\right) \tag{82}$$

The contribution from terms of the second type is:

$$\alpha^2\beta^2\sigma^4 P\sum_{m=1}^{r}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)^\top\nabla_{\boldsymbol{\theta}_m}L_j(\Theta) \tag{83}$$

Term $\textcircled{ii}$. Here, two types of terms have non-zero expectations: we have r terms of type $\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_s^\top$ and r terms of type $\boldsymbol{\epsilon}_s\boldsymbol{\epsilon}_s^\top\boldsymbol{\epsilon}_m\boldsymbol{\epsilon}_m^\top$. The total contribution of both terms is:

$$\alpha^2\beta^2\sigma^4\sum_{m=1}^{r}\left(\sum_{m'=1}^{i}\nabla_{\boldsymbol{\theta}_{m'}}L_i(\Theta)\right)\nabla_{\boldsymbol{\theta}_m}L_j(\Theta)+\alpha^2\beta^2\sum_{m=1}^{r}\nabla_{\boldsymbol{\theta}_m}L_i(\Theta)^\top\left(\sum_{m'=1}^{j}\nabla_{\boldsymbol{\theta}_{m'}}L_j(\Theta)\right) \tag{84}$$

Term $\textcircled{iii}$. Term $\textcircled{iii}$ is symmetrical to term $\textcircled{ii}$, swapping i for j. Thus, its contribution is identical to that of term $\textcircled{ii}$:

$$\alpha^2\beta^2\sigma^4 \sum_{m=1}^{r} \left(\sum_{m'=1}^{i} \nabla_{\boldsymbol{\theta}_{m'}} L_i(\Theta) \right) \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) + \alpha^2\beta^2 \sum_{m=1}^{r} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \left(\sum_{m'=1}^{j} \nabla_{\boldsymbol{\theta}_{m'}} L_j(\Theta) \right) \tag{85}$$

Term $\textcircled{iv}$. Here, we have three types of terms with non-zero expectation: r terms of the form $\boldsymbol{\epsilon}_s \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_s^\top$, r terms of the form $\boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_m^\top$, and $r(r-1)$ terms of the form $\boldsymbol{\epsilon}_n \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_n^\top$ where $n \neq m$.

The contribution from the first type, $\boldsymbol{\epsilon}_s \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_s^\top$, is:

$$\alpha^2\beta^2\sigma^4 P \sum_{m=1}^{r} \left(\sum_{m'=1}^{i} \nabla_{\boldsymbol{\theta}_{m'}} L_i(\Theta) \right)^\top \left(\sum_{n'=1}^{j} \nabla_{\boldsymbol{\theta}_{n'}} L_j(\Theta) \right) \tag{86}$$

The contribution from the second type, $\boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_m^\top$, is:

$$\beta^4\sigma^4(P+2) \sum_{m=1}^{r} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) \tag{87}$$

The contribution from the third type, $\boldsymbol{\epsilon}_n \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_n^\top$ where $n \neq m$, is:

$$\beta^4\sigma^4 P \sum_{n=1}^{r} \left(\sum_{m \leq r, m \neq n} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) \right) \tag{88}$$

Term $\textcircled{v}$. Here, we have two types of terms with non-zero expectation: $\boldsymbol{\epsilon}_n \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_n \boldsymbol{\epsilon}_m^\top$ and $\boldsymbol{\epsilon}_m \boldsymbol{\epsilon}_m^\top \boldsymbol{\epsilon}_n \boldsymbol{\epsilon}_n^\top$. The contribution of these terms is:

$$\beta^4\sigma^4 \sum_{m \leq i, n \leq j, m \neq n} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_n} L_j(\Theta) + \beta^4\sigma^4 \sum_{m \leq i, n \leq j, m \neq n} \nabla_{\boldsymbol{\theta}_n} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) \tag{89}$$

Combining Terms. Overall, we have T terms of type ⓐ, e.g., with structure $\boldsymbol{g}_i^\top \boldsymbol{v}_i (\alpha \boldsymbol{\epsilon}_s + \beta \boldsymbol{\xi}_i)^\top (\alpha \boldsymbol{\epsilon}_s + \beta \boldsymbol{\xi}_i) \boldsymbol{v}_i^\top \boldsymbol{g}_i$, with total contribution:

$$\begin{aligned}
&\sum_{i=1}^{T} \left(\alpha^4\sigma^4(P+2) \left(\sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \right)^\top \left(\sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \right) \right. && (90)\\
&\quad + \alpha^2\beta^2\sigma^4 P \sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) + 4\alpha^2\beta^2\sigma^4 \left(\sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \right)^\top \left(\sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \right) && (91)\\
&\quad + \alpha^2\beta^2\sigma^4 P i \left(\sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \right)^\top \left(\sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \right) + \beta^4\sigma^4(P+2) \sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) && (92)\\
&\quad \left. + \beta^4\sigma^4 P \sum_{n=1}^{i} \left(\sum_{m \leq i, m \neq n} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_n} L_i(\Theta) \right) + 2\beta^4\sigma^4 \sum_{m \leq i, n \leq i, m \neq n} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_n} L_i(\Theta) \right) && (93)
\end{aligned}$$

In addition, we have several terms of type ⓑ, e.g., with structure $\boldsymbol{g}_i^\top \boldsymbol{v}_i (\alpha \boldsymbol{\epsilon}_s + \beta \boldsymbol{\xi}_i)^\top (\alpha \boldsymbol{\epsilon}_s + \beta \boldsymbol{\xi}_j)^\top \boldsymbol{v}_j^\top \boldsymbol{g}_j$, with total contribution:

$$
\begin{aligned}
&\sum_{i \neq j} \left(\alpha^4 \sigma^4 (P+2) \left(\sum_{m=1}^{i} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta) \right)^\top \left(\sum_{n=1}^{j} \nabla_{\boldsymbol{\theta}_n} L_j(\Theta) \right) + \alpha^2 \beta^2 \sigma^4 P \sum_{m=1}^{r} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) \right. && (94) \\
&\quad + 2\alpha^2 \beta^2 \sigma^4 \sum_{m=1}^{r} \left(\sum_{m'=1}^{i} \nabla_{\boldsymbol{\theta}_{m'}} L_i(\Theta) \right) \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) + 2\alpha^2 \beta^2 \sum_{m=1}^{r} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \left(\sum_{m'=1}^{j} \nabla_{\boldsymbol{\theta}_{m'}} L_j(\Theta) \right) && (95) \\
&\quad + \alpha^2 \beta^2 \sigma^4 P \sum_{m=1}^{r} \left(\sum_{m'=1}^{i} \nabla_{\boldsymbol{\theta}_{m'}} L_i(\Theta) \right)^\top \left(\sum_{n'=1}^{j} \nabla_{\boldsymbol{\theta}_{n'}} L_j(\Theta) \right) && (96) \\
&\quad + \beta^4 \sigma^4 (P+2) \sum_{m=1}^{r} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) + \beta^4 \sigma^4 P \sum_{n=1}^{r} \left(\sum_{m \leq r, m \neq n} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) \right) && (97) \\
&\quad \left. + \beta^4 \sigma^4 \sum_{m \leq i, n \leq j, m \neq n} \nabla_{\boldsymbol{\theta}_m} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_n} L_j(\Theta) + \beta^4 \sigma^4 \sum_{m \leq i, n \leq j, m \neq n} \nabla_{\boldsymbol{\theta}_n} L_i(\Theta)^\top \nabla_{\boldsymbol{\theta}_m} L_j(\Theta) \right) && (98)
\end{aligned}
$$

Recovering the Variance of ES-Single and PES. Setting $\alpha = 0, \beta = 1$ recovers the PES variance, while setting $\alpha = 1, \beta = 0$ recovers the variance of ES-Single. Note that most of the terms in the variance of this generalized estimator include coefficient β; thus, intuitively one would expect that setting $\beta = 0$ (for ES-Single) has reduced variance relative to PES.

H. Code

Code Listing 1 provides a self-contained JAX implementation of the ES-Single gradient estimator, that reproduces the result in Figure 15.

Listing 1. Self-contained implementation of ES-Single in JAX, for the 2D regression problem in Figure 15.

```python
from functools import partial
import jax
import jax.numpy as jnp

import optax

def loss(x):
   """Inner loss."""
   return jnp.sqrt(x[0]**2 + 5) - jnp.sqrt(5) + jnp.sin(x[1])**2 * \
          jnp.exp(-5*x[0]**2) + 0.25*jnp.abs(x[1] - 100)

# Gradient of inner loss
loss_grad = jax.grad(loss)

def update(state, i):
   """Performs a single inner problem update, e.g., a single unroll step.
   """
   (L, x, theta, t_curr, T, K) = state
   lr = jnp.exp(theta[0]) * (T - t_curr) / T + jnp.exp(theta[1]) * t_curr / T
   x = x - lr * loss_grad(x)
   L += loss(x) * (t_curr < T)
   t_curr += 1
   return (L, x, theta, t_curr, T, K), x

@partial(jax.jit, static_argnames=('T', 'K'))
def unroll(x_init, theta, t0, T, K):
   """Unroll the inner problem for K steps.

   Args:
     x_init: the initial state for the unroll
     theta: a 2-dimensional array of outer parameters (log_init_lr, log_final_lr)
     t0: initial time step to unroll from
     T: maximum number of steps for the inner problem
     K: number of steps to unroll

   Returns:
     L: the loss resulting from the unroll
     x_curr: the updated state at the end of the unroll
   """
   L = 0.0
   initial_state = (L, x_init, theta, t0, T, K)
   state, outputs = jax.lax.scan(update, initial_state, None, length=K)
   (L, x_curr, theta, t_curr, T, K) = state
   return L, x_curr

@partial(jax.jit, static_argnames=(’T’, ’K’, ’sigma’, ’N’))
def es_single_grad(key, xs, theta, t0, T, K, sigma, N):
   """Compute ES-Single gradient estimate.

   Args:
     key: JAX PRNG key
     xs: Nx2 array of particles/states to be updated
     theta: a 2-dimensional array of outer parameters (log_init_lr, log_final_lr)
     t0: initial time step for the current unroll
     T: maximum number of steps for the inner problem
     K: truncation length for the unroll
     sigma: standard deviation of the Gaussian perturbations
     N: number of perturbations (as N//2 antithetic pairs)

   Returns:
     theta_grad: ES-Single gradient estimate
     xs: Nx2 array of updates particles/states
   """
   # Generate antithetic perturbations
   pos_perts = jax.random.normal(key, (N//2, theta.shape[0])) * sigma # Antithetic pos
   neg_perts = -pos_perts # Antithetic neg
   perts = jnp.concatenate([pos_perts, neg_perts], axis=0)

   # Unroll the inner problem for K steps using the antithetic perturbations of theta
   L, xs = jax.vmap(unroll, in_axes=(0,0,None,None,None))(xs, theta + perts, t0, T, K)
   # Compute the ES-Single gradient estimate
   theta_grad = jnp.mean(perts * L.reshape(-1, 1) / (sigma**2), axis=0)
   return theta_grad, xs

T = 100   # Total inner problem length
K = 10    # Truncation length for partial unrolls
N = 100   # Number of particles in total (N//2 antithetic pairs)
sigma = 0.1 # Standard deviation of perturbations

t = 0
theta = jnp.log(jnp.array([0.01, 0.01]))
x = jnp.array([1.0, 1.0])
xs = jnp.ones((N, 2)) * jnp.array([1.0, 1.0])

optimizer = optax.adam(1e-2)
opt_state = optimizer.init(theta)

key = jax.random.PRNGKey(3)
for i in range(10000):
   if t >= T:
      # Reset the inner problem: the inner iteration, inner parameters, and random key
      key, skey = jax.random.split(key)
      t = 0
      xs = jnp.ones((N, 2)) * jnp.array([1.0, 1.0])
      x = jnp.array([1.0, 1.0])
```

```
theta_grad, xs = es_single_grad(key, xs, theta, t, T, K, sigma, N)

updates, opt_state = optimizer.update(theta_grad, opt_state)
theta = optax.apply_updates(theta, updates)

t += K

if i % 100 == 0:
   # Run a full unroll for evaluation
   L, _ = unroll(jnp.array([1.0, 1.0]), theta, 0, T, T)
   print(i, jnp.exp(theta), theta_grad, L)
```