Unbiased Gradient Estimation in Unrolled Computation Graphs with Persistent Evolution Strategies

Supplementary Material

Paul Vicol Luke Metz Jascha Sohl-Dickstein

This appendix is structured as follows:

- In Section A we give an overview of the notation used in this paper.
- In Section B we provide a table comparing several hyperparameter optimization approaches.
- In Section C we provide experimental details.
- In Section D we discuss telescoping sums as a way to target the final loss rather than the sum of losses as the meta-objective.
- In Section E we provide a derivation of the PES estimator.
- In Section F we prove that PES is unbiased.
- In Section G we derive the variance of the PES estimator.
- In Section H we derive a variant of the PES estimator that incorporates the analytic gradient from the most recent partial unroll to reduce variance.
- In Section I we show the connection between PES and the framework for gradient estimation in stochastic computation graphs introduced in Schulman et al. (2015).
- In Section J we show derivations and compute/memory costs of the methods in Table 1.
- In Section K we provide diagrammatic representations of the ES and PES algorithms.
- In Section L we provide an ablation study over the truncation length and number of particles for PES.
- In Section M we provide simplified code to implement PES in JAX (Bradbury et al., 2018).

A. Notation

Table 2 summarizes the notation used in this paper.

Symbol	Meaning
ES	Evolution strategies
PES	Persistent evolution strategies
(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 $\boldsymbol{\theta}_t = \boldsymbol{\theta}, \forall t$
Θ	A matrix whose rows are the parameters at each timestep, $\Theta = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T)^\top$
$\boldsymbol{s}_t$	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
$\boldsymbol{\epsilon}$	A matrix whose rows are the perturbations at each timestep, $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}_1, \dots, \boldsymbol{\epsilon}_T)^\top$
$\boldsymbol{\xi}_t$	The sum of PES perturbations up to time t, $\boldsymbol{\xi}_t = \boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_t$
$L_t(\Theta)$	The loss at timestep t, $L_t(\Theta) = L_t(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t)$
$L(\boldsymbol{\theta})$, $L(\Theta)$	The total loss, $L(\boldsymbol{\theta}) = L(\Theta) = \sum_{t=1}^{T} L_t(\Theta) = \sum_{t=1}^{T} L_t(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t)$
$\boldsymbol{g}_t$	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)
$\hat{\boldsymbol{g}}^{\text{PES}}$	The PES gradient estimate (with Monte-Carlo sampling)
$\hat{\boldsymbol{g}}^{\text{PES-A}}$	The antithetic PES gradient estimate (with Monte-Carlo sampling)
$g\,(t, \tau)$	Shorthand for $\frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}_\tau}$, used in variance expressions
$\boldsymbol{p}_t$	Shorthand for $\frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}_t}$, used in the PES-Analytic estimate
$\otimes$	Kronecker product
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 2. **Table of notation, defining the terms we use in this paper.**

B. Hyperparameter Optimization Methods

Table 3 presents a comparison of several hyperparameter optimization approaches. We distinguish between black-box, gray-box, and gradient-based approaches, and focus our comparison on whether each method can tune optimization hyperparameters, regularization hyperparameters, and discrete hyperparameters, as well as whether the method requires multiple runs through the inner problem or is online (operating within the timespan of a single inner problem), and whether the method is unbiased, meaning that it will eventually converge to the optimal hyperparameters.

Method	Type	Parallel	Tune Opt	Tune Reg	Tune Discrete	Online (1 Run)	Unbiased
Grid/Random (Bergstra & Bengio, 2012)	■	✓	✓	✓	✓	✗	✓
BayesOpt (Snoek et al., 2012)	■	✓	✓	✓	✓	✗	✓
SHAC (Kumar et al., 2018)	■	✓	✓	✓	✓	✗	✗
Freeze-Thaw BO (Swersky et al., 2014)	■	✓	✓	✓	✓	✗	✓
Full ES	■	✓	✓	✓	✓	✗	✓
PBT (Jaderberg et al., 2017)	■	✓	✓	✓	✓	✓	✗
Succ. Halving (Jamieson & Talwalkar, 2016)	■	✓	✓	✓	✓	✗	✗
Hyperband (Li et al., 2017)	■	✓	✓	✓	✓	✗	✗
Trunc. Unroll (Maclaurin et al., 2015)	∇	✗	✓	✓	✗	✗	✗
STN (MacKay et al., 2019)	∇	✗	✗	✓	✓	✓	✗
IFT (Lorraine et al., 2020)	∇	✗	✗	✓	✗	✓	✓
HD (Baydin et al., 2017)	∇	✗	✓	✗	✗	✓	✗
MARTHE (Donini et al., 2019)	∇	✗	✓	✓	✗	✓	✗
RTHO (Franceschi et al., 2017)	∇	✗	✓	✓	✗	✓	✓
PES (Ours)	■	✓	✓	✓	✓	✗	✓
PES+Analytic (Ours)	∇	✓	✓	✓	✗	✗	✓

Table 3. **Comparison between hyperparameter optimization approaches.** ■ denotes *black-box*, ■ denotes *gray-box*, and ∇ denotes *gradient-based* approaches.

C. Experiment Details

In this section, we provide details for the experiments from Section 5.

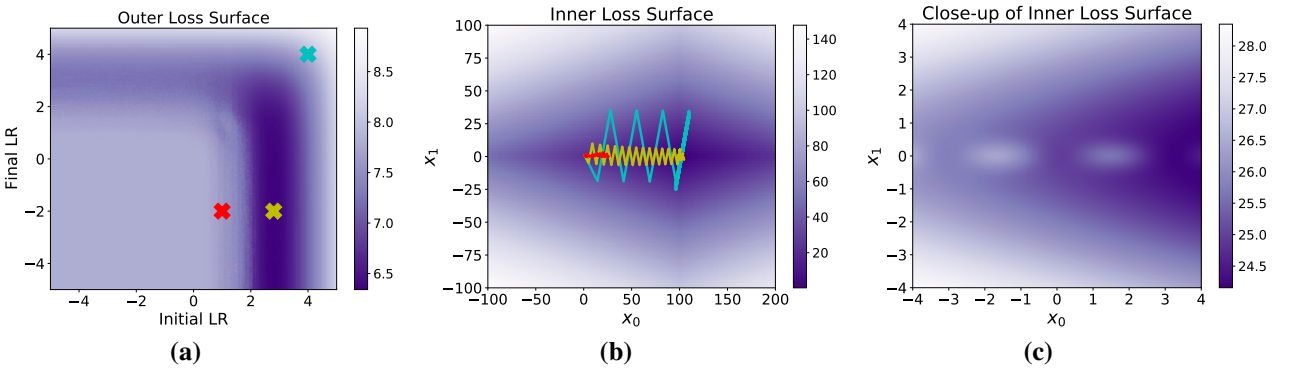
Computing Infrastructure. All experiments except for learned optimizer training were run on NVIDIA P100 GPUs (using only a single GPU per experiment). The learned optimizer experiment in Section 5.2 was trained on 8 TPUv2 cores; we used asynchronous multi-TPU training for convenience, not necessity (these experiments could be run on a single GPU if desired).

C.1. 2D Toy Regression

The inner objective is a toy 2D function defined as:

$$f(x_0, x_1) = \sqrt{x_0^2 + 5} - \sqrt{5} + \sin^2(x_1)\exp(-5x_0^2) + 0.25|x_1 - 100| \tag{12}$$

This was manually designed to be a challenging problem for any meta-optimization method that suffers from truncation bias. In Figure 10 we visualize the outer loss surface (aka the meta-loss surface) and the inner loss surface for this task; we show the optimization trajectories on the inner loss surface corresponding to three different choices of optimization hyperparameters (shown by color-coded markers).



(a) **(b)** **(c)**

Figure 10. **Optimization landscape for the toy 2D regression problem.** **(a)** The outer loss (e.g. meta-loss) surface, showing the meta-objective values (e.g. the sum of losses over the inner optimization trajectory) for different settings of the two hyperparameters controlling the initial and final (log) learning rates of a linear decay schedule; **(b)** the inner loss surface, showing color-coded optimization trajectories corresponding to the hyperparameters highlighted in (a); **(c)** a close-up of the inner loss surface in the region where the parameters (x_0, x_1) are initialized at the start of the inner problem.

In our experiments, the total inner problem length was $T = 100$, and we used truncated unrolls of length $K = 10$. For ES and PES, we used perturbation variance $\sigma^2 = 1$, and 100 particles (50 antithetic pairs). We used Adam with learning rate 1e-2 as the outer optimizer for all methods (TBPTT, RTRL, UORO, ES, and PES).

C.2. Influence Balancing

The influence balancing task considers learning a scalar parameter $\theta \in \mathbb{R}$ that governs the evolution of the following unrolled system:

$$\boldsymbol{s}_{t+1} = A\boldsymbol{s}_t + (\underbrace{\theta, \ldots, \theta}_{p \text{ positive}}, \underbrace{-\theta, \ldots, -\theta}_{n-p \text{ negative}})^\top \tag{13}$$

where A is a fixed $n \times n$ matrix with $A_{i,i} = 0.5$, $A_{i,i+1} = 0.5$ and 0 everywhere else. The vector on the right hand side consists of θ tiled n times, with p positive and $n - p$ negative copies. In our experiments, we used $n = 23$ and $p = 10$. The loss at each step is regression on the first index in the state vector $\boldsymbol{s}_t$:

$$L_t = \frac{1}{2}(\boldsymbol{s}_t^0 - 1)^2 \tag{14}$$

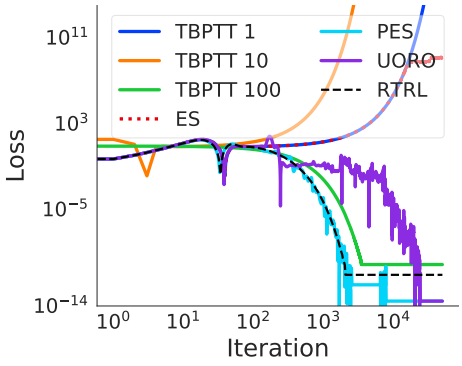


Figure 11. Longer run of influence balancing, with log-scaled x-axis.

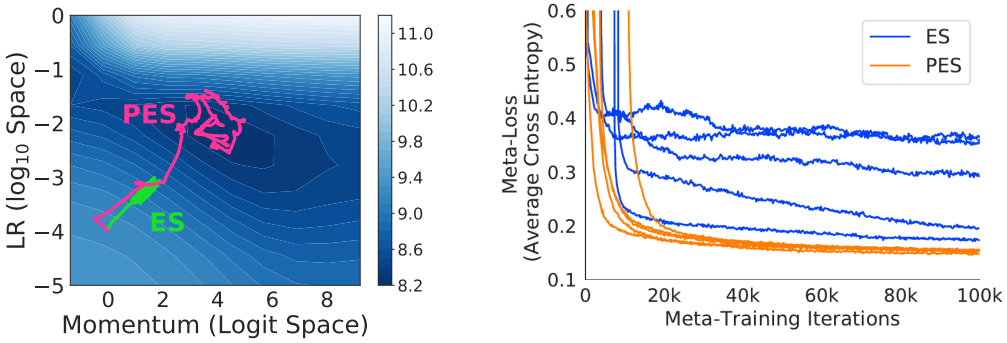
For the influence balancing experiment, we used $n = 23$ with 10 positive and 13 negative θ's. The state was initialized to a vector of ones, $\boldsymbol{s}_0 = \mathbf{1}$, and θ was initialized to 0.5. We used gradient descent for optimization, with learning rate 1e-4. We did not use learning rate decay as was used in (Tallec & Ollivier, 2017a), as we did not find this to be necessary for convergence. For ES and PES we used perturbation scale $\sigma = 0.1$ and 10^3 particles.

C.3. MNIST Experiments

MNIST Meta-Optimization. Following Wu et al. (2018), we used a two-layer MLP with 100 hidden units per layer and ReLU activations and the learning rate schedule parameterization $\alpha_t = \frac{\theta_0}{\left(1+\frac{t}{Q}\right)^{\theta_1}}$, where α_t is the learning rate at step t, θ_0 is the initial learning rate, θ_1 is the decay factor, and Q is a constant fixed to 5000. This schedule is used for SGD with fixed momentum 0.9. We used mini-batches of size 100. The full unrolled inner problem consists of $T = 5000$ optimization steps, and we used vanilla ES and PES with truncation lengths $K \in \{10, 100\}$, yielding 500 and 50 unrolls per inner problem. The meta-objective is the sum of training softmax cross-entropy losses over the inner optimization trajectory. We used Adam as the outer-optimizer, and for each method (ES and PES), we performed a grid search over the outer-learning rates $\{0.01, 0.03, 0.1\}$ to find the most stable and fastest-converging setups. For both ES and PES, we used perturbation standard deviation $\sigma = 0.1$, and 1000 particles (500 antithetic pairs).

Tuning Many Hyperparameters & Comparison to Random Search. For this experiment, we used a four-hidden-layer MLP with 100 hidden units per layer and ReLU nonlinearities (architecture $\text{Linear}(784, 100) \rightarrow \text{ReLU} \rightarrow \text{Linear}(100, 100) \rightarrow \text{ReLU} \rightarrow \text{Linear}(100, 100) \rightarrow \text{ReLU} \rightarrow \text{Linear}(100, 100) \rightarrow \text{ReLU} \rightarrow \text{Linear}(100, 10)$), trained on mini-batches of size 100. We tuned a separate learning rate and momentum coefficient for each weight matrix and bias in the network: we have 5 layers in total (including the output layer), each layer has a weight and bias, and each weight/bias has 2 tuned hyperparameters, yielding 20 total hyperparameters for this task. The meta-objective was the sum of training losses over the inner optimization trajectory. The total inner problem length was $T = 1000$, and for ES and PES we used truncations of length $K = 10$, yielding 100 unrolls per inner problem. For random search, we sampled learning rates uniformly at random in log-space, with range $(e^{-8}, e^{0}) = (\sim \text{3e-4}, 1)$; we sampled the pre-transformed momentum coefficients uniformly at random in the range $(-10, 10)$ then transformed them by a sigmoid, so that the range corresponds to the constrained space $(\sim 4.5\text{e-}5,\ 0.9999)$. These values were selected to be uninformative, while remaining stable in most settings. For ES and PES, we initialized each learning rate uniformly at random in log space in the range $(\text{1e-4}, \text{1e-2})$; we initialized each momentum coefficient uniformly at random in the pre-transformed space corresponding to the sigmoid-transformed range $(0.01, 0.9)$. These ranges are slightly smaller than the ones used for random search in order to maintain meta-optimization stability; note from Figure 9 that the performance of both ES and PES is initially poor (prior to meta-optimization), indicating that these ranges for random initialization do not increase their performance compared to random search, and thus the improvement for PES is primarily due to its adaptation of the hyperparameters. For ES and PES, we used perturbation standard deviation $\sigma = 0.1$, $N = 2$ particles (1 antithetic pair), and Adam with learning rate 0.01 for outer optimization. We ran each method four times with different random seeds, and plotted the mean performance, with the min and max shown by the shaded regions in Figure 9. We measured the best meta-objective value achieved so far during meta-optimization, as a function of *total compute*, which takes into account the number of inner iterations performed, as well as the number of parallel workers (or particles); total compute corresponds to the product of inner iterations and the number of workers.

Additional Hyperparameter Optimization and Learned Optimizer Experiments. In Figure 12a, we tune hyperparameters for a 1.6M parameter ResNet on CIFAR-10 using ES and PES with $T = 5000$, $K = 20$, and $N = 4$, targeting the sum of validation losses. In Figure 12b, we train a learned optimizer on MNIST (similarly to Metz et al. (2019)). We use the same configuration as described in section 5.2 but target a 2 hidden layer, 128 unit MLP trained on MNIST.

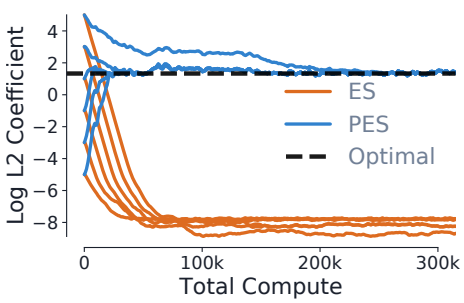


(a) Tuning LR & momentum for a Myrtle.ai ResNet on CIFAR-10.

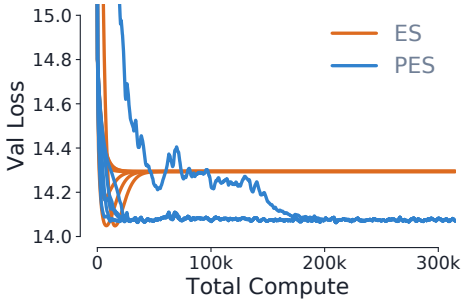
(b) Learned optimizer trained on MNIST.

Figure 12. CIFAR-10 experiment for hyperparameter optimization and MNIST experiment for learned optimizer training.

Tuning Regularization for UCI Regression. Here we show that truncation bias can also arise for regularization hyperparameters such as the L_2 regularization coefficient. We tune L_2 regularization for linear regression on the Yacht data from the UCI collection (Asuncion & Newman, 2007). We found the optimal L_2 coefficient using a fine-trained grid search. In Figure 13 we compare meta-optimization using ES and PES, starting from different initial L_2 coefficients; PES robustly converges to the correct solution in all cases. We used $\sigma = 0.01$, $K = 1$, and $N = 4$ for both ES and PES.



(a) Meta-optimization trajectories using ES and PES from different initial L_2 coefficients.



(b) Validation losses using ES and PES for differnet initial L_2 values.

Figure 13. Using ES and PES to tune the L_2 regularization coefficient for linear regression on the UCI Yacht dataset.

C.4. Continuous Control Details

We used OpenAI Gym[4] to interface with MuJoCo. In our implementation, each antithetic pair shares a MuJoCo environment state, which is different between different antithetic pairs. The environment state is reset to the same point before running the partial unrolls of each particle in a pair, to control for randomness (e.g., the antithetic perturbations are evaluated starting from a common state). As is standard for MuJoCo environments, the length of a full episode is $T = 1000$; we ran full-unroll ES with $K = 1000$, and we used partial unrolls of length $K = 100$ for truncated ES and PES. Following Mania et al. (2018), we used a linear policy initialized as all 0s (the linear policy is a single weight matrix with no bias term). Also following Mania et al. (2018), we divided the rewards by their standard deviation (computed using the aggregated rewards from all antithetic pairs) before computing the ES/PES gradient estimates. We did not use state normalization, nor did we perform any heuristic selection of a subset of the best sampled perturbation directions (as used in the ARS V2 approach of Mania et al. (2018)).

D. Telescoping Sums

If we wish to target the final loss L_T as the meta-objective, we can define $p_t = L_t - L_{t-1}$, where $L_{-1} \equiv 0$. This yields the telescoping sum:

$$\sum_{t=0}^{T} p_t = (\cancel{L_0} - L_{-1}) + (\cancel{L_1} - \cancel{L_0}) + (\cancel{L_2} - \cancel{L_1}) + \cdots + (\cancel{L_{T-1}} - \cancel{L_{T-2}}) + (L_T - \cancel{L_{T-1}}) = L_T \tag{15}$$

Targeting the final loss encourages different behavior than targeting the sum or average of the losses. Targeting the sum of losses encourages fast convergence (small $\sum_t L_t$), but not necessarily the smallest final loss L_T, while targeting the final loss encourages finding the smallest L_T potentially at the expense of slower convergence (larger $\sum_t L_t$).

We performed an experiment using telescoping sums to target the final training loss, optimizing an exponential LR schedule for an MLP on FashionMNIST with $T = 5000$, $K = 20$, $N = 100$ (Figure 14). Due to the computational expense of evaluating the loss on the full training set to obtain L_t at each partial unroll, we selected a random minibatch at the start of each inner problem, which was kept fixed for the loss evaluations for that inner problem.

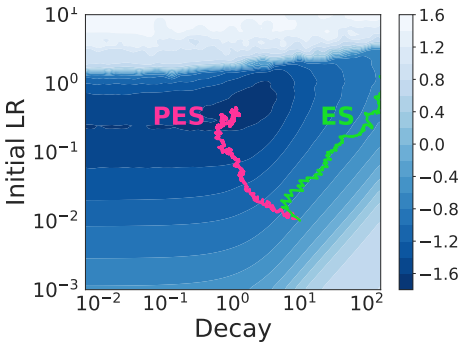


Figure 14. Telescoping sum for FashionMNIST final training loss (colors show log-final-loss).

E. Derivation of Persistent Evolution Strategies

Here we derive the PES estimator. The derivation here closely follows that in the text body, but shows additional intermediate steps in several places in the derivation. Also see Appendix I for an alternate derivation using stochastic computation

[4] https://github.com/openai/gym

graphs (Schulman et al., 2015).

E.1. Notation

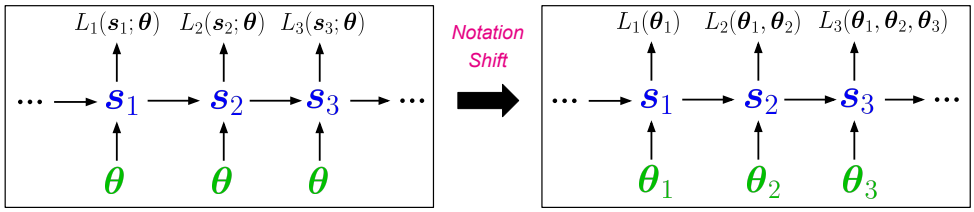


Figure 15. Shift in notation, dropping the dependence on $\boldsymbol{s}_t$ and explicitly including the dependence on each $\boldsymbol{\theta}_t$.

Unrolled computation graphs (as illustrated in Figure 1) depend on shared parameters $\boldsymbol{\theta}$ at every timestep. In order to account for how these contribute to the overall gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$, we use subscripts $\boldsymbol{\theta}_t$ to distinguish between applications of $\boldsymbol{\theta}$ at different steps, where $\boldsymbol{\theta}_t = \boldsymbol{\theta}, \forall t$ (see Figure 15). We further define $\Theta = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T)^\top$, which is a matrix with the per-timestep $\boldsymbol{\theta}_t$ as its rows. For notational simplicity in the following derivation, we drop the dependence on $\boldsymbol{s}_t$ and explicitly include the dependence on each $\boldsymbol{\theta}_t$, writing $L_t(\boldsymbol{s}_t;\boldsymbol{\theta})$ as either $L_t(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t)$ or simply $L_t(\Theta)$, with an implicit initial state $\boldsymbol{s}_0$. Thus, $L(\boldsymbol{\theta}) = \sum_{t=1}^T L_t(\boldsymbol{s}_t;\boldsymbol{\theta}) = \sum_{t=1}^T L_t(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_t) = \sum_{t=1}^T L_t(\Theta)$.

E.2. PES is ES Over the Parameters at Each Unroll Step

We wish to compute the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ of the total loss over all unrolls. We begin by writing this gradient in terms of the full gradient $\frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)} \in \mathbb{R}^{PT \times 1}$, where P is the number of parameters, and T is the total number of unrolls, and then using ES to approximate $\frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)}$. First, note that we can write:

$$\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \frac{dL(\Theta)}{d\boldsymbol{\theta}} = \frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_1} \cancelto{1}{\frac{d\boldsymbol{\theta}_1}{d\boldsymbol{\theta}}} + \frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_2} \cancelto{1}{\frac{d\boldsymbol{\theta}_2}{d\boldsymbol{\theta}}} + \cdots + \frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_T} \cancelto{1}{\frac{d\boldsymbol{\theta}_T}{d\boldsymbol{\theta}}} = \sum_{\tau=1}^{T} \frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_\tau} = \left(\mathbf{I} \otimes \mathbf{1}^\top\right) \frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)}$$

where $\otimes$ denotes the Kronecker product, $\mathbf{I}$ has dimension $P \times P$, $\mathbf{1}^\top$ has dimension $1 \times T$, and thus $\mathbf{I} \otimes \mathbf{1}^\top$ has dimension $P \times PT$. Note that because $\frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)}$ has dimension $PT \times 1$, the product $\left(\mathbf{I} \otimes \mathbf{1}^\top\right) \frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)}$ will be $P \times 1$. Next, we will apply ES to approximate the last RHS expression above:

$$\begin{aligned}
\frac{dL(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \approx \boldsymbol{g}^{\text{PES}} &= \left(\mathbf{I} \otimes \mathbf{1}^\top\right) \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{1}{\sigma^2} \operatorname{vec}(\boldsymbol{\epsilon})\, L(\Theta + \boldsymbol{\epsilon})\right] \\
&= \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}} \left[\left(\mathbf{I} \otimes \mathbf{1}^\top\right) \operatorname{vec}(\boldsymbol{\epsilon})\, L(\Theta + \boldsymbol{\epsilon})\right] \\
&= \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}} \left[\left(\sum_{\tau=1}^{T} \boldsymbol{\epsilon}_\tau\right) L(\Theta + \boldsymbol{\epsilon})\right],
\end{aligned}$$

where $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}_1, \ldots, \boldsymbol{\epsilon}_T)^\top$ is a matrix of perturbations $\boldsymbol{\epsilon}_t$ to be added to the $\boldsymbol{\theta}_t$ at each timestep and the expectation is over entries in $\boldsymbol{\epsilon}$ drawn from an iid Gaussian with variance σ^2. This ES approximation is an unbiased estimator of the gradient of the Gaussian-smoothed objective $\mathbb{E}_{\boldsymbol{\epsilon}}[L(\Theta + \boldsymbol{\epsilon})]$.

We next show that $\boldsymbol{g}^{\text{PES}}$ decomposes into a sum of sequential gradient estimates,

$$\begin{aligned}
\boldsymbol{g}^{\text{PES}} &= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau=1}^{T}\boldsymbol{\epsilon}_\tau\right)L\left(\Theta+\boldsymbol{\epsilon}\right)\right] \\
&= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau=1}^{T}\boldsymbol{\epsilon}_\tau\right)\sum_{t=1}^{T}L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right] \\
&= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\left(\sum_{\tau=1}^{T}\boldsymbol{\epsilon}_\tau\right)L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right] && (16) \\
&= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\left(\sum_{\tau=1}^{t}\boldsymbol{\epsilon}_\tau\right)L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right] && (17) \\
&= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\boldsymbol{\xi}_t L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right] && (18) \\
&= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\hat{\boldsymbol{g}}_{t,\boldsymbol{\epsilon}}^{\text{PES}}\right], && (19) \\
\hat{\boldsymbol{g}}_{t,\boldsymbol{\epsilon}}^{\text{PES}} &= \frac{1}{\sigma^2}\boldsymbol{\xi}_t L_t\left(\Theta+\boldsymbol{\epsilon}\right) = \frac{1}{\sigma^2}\boldsymbol{\xi}_t L_t\left(\boldsymbol{\theta}_1+\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\theta}_t+\boldsymbol{\epsilon}_t\right). && (20)
\end{aligned}$$

where $\boldsymbol{\xi}_t = \sum_{\tau=1}^{t}\boldsymbol{\epsilon}_\tau$, Equation 17 relies on $L_t\left(\cdot\right)$ being independent of $\boldsymbol{\epsilon}_\tau$ for $\tau > t$, and Equation 20 similarly relies on $L_t\left(\cdot\right)$ only being a function of $\boldsymbol{\theta}_\tau$ for $\tau \leq t$.

The PES estimator consists of Monte Carlo estimates of Equation 19,

$$\hat{\boldsymbol{g}}^{\text{PES}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\hat{\boldsymbol{g}}_{t,\boldsymbol{\epsilon}^{(i)}}^{\text{PES}} \tag{21}$$

where $\boldsymbol{\epsilon}^{(i)}$ are samples of $\boldsymbol{\epsilon}$, and N is the number of Monte Carlo samples. Gradient estimates at each time step can be evaluated sequentially, and used to perform SGD.

Concrete Example. To illustrate how the expressions in the derivation above yield the desired gradient estimate, here we provide a concrete example using two-dimensional $\boldsymbol{\theta}$ with three steps of unrolling. The matrix Θ is:

$$\Theta = \begin{bmatrix} —— \boldsymbol{\theta}_1 —— \\ —— \boldsymbol{\theta}_2 —— \\ —— \boldsymbol{\theta}_3 —— \end{bmatrix} = \begin{bmatrix} \theta_1^{(1)} & \theta_1^{(2)} \\ \theta_2^{(1)} & \theta_2^{(2)} \\ \theta_3^{(1)} & \theta_3^{(2)} \end{bmatrix}$$

The vectorized matrix $\text{vec}\left(\Theta\right)$ and gradient $\frac{\partial L(\Theta)}{\partial\,\text{vec}(\Theta)}$ are as follows:

$$\text{vec}(\Theta) = \begin{bmatrix} \theta_1^{(1)} \\ \theta_2^{(1)} \\ \theta_3^{(1)} \\ \theta_1^{(2)} \\ \theta_2^{(2)} \\ \theta_3^{(2)} \end{bmatrix} \qquad \frac{\partial L(\Theta)}{\partial \text{vec}(\Theta)} = \begin{bmatrix} \frac{\partial L(\Theta)}{\partial \theta_1^{(1)}} \\ \frac{\partial L(\Theta)}{\partial \theta_2^{(1)}} \\ \frac{\partial L(\Theta)}{\partial \theta_3^{(1)}} \\ \frac{\partial L(\Theta)}{\partial \theta_1^{(2)}} \\ \frac{\partial L(\Theta)}{\partial \theta_2^{(2)}} \\ \frac{\partial L(\Theta)}{\partial \theta_3^{(2)}} \end{bmatrix}$$

The Kronecker product is: $\mathbf{I} \otimes \mathbf{1}^\top = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \otimes \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$. Thus, we have:

$$(\mathbf{I} \otimes \mathbf{1}^\top) \frac{\partial L(\Theta)}{\partial \text{vec}(\Theta)} = \begin{bmatrix} \frac{\partial L(\Theta)}{\partial \theta_1^{(1)}} + \frac{\partial L(\Theta)}{\partial \theta_2^{(1)}} + \frac{\partial L(\Theta)}{\partial \theta_3^{(1)}} \\ \frac{\partial L(\Theta)}{\partial \theta_1^{(2)}} + \frac{\partial L(\Theta)}{\partial \theta_2^{(2)}} + \frac{\partial L(\Theta)}{\partial \theta_3^{(2)}} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial L(\Theta)}{\partial \theta_1^{(1)}} \\ \frac{\partial L(\Theta)}{\partial \theta_1^{(2)}} \end{bmatrix}}_{\frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_1}} + \underbrace{\begin{bmatrix} \frac{\partial L(\Theta)}{\partial \theta_2^{(1)}} \\ \frac{\partial L(\Theta)}{\partial \theta_2^{(2)}} \end{bmatrix}}_{\frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_2}} + \underbrace{\begin{bmatrix} \frac{\partial L(\Theta)}{\partial \theta_3^{(1)}} \\ \frac{\partial L(\Theta)}{\partial \theta_3^{(2)}} \end{bmatrix}}_{\frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_3}} = \sum_{\tau=1}^{T} \frac{\partial L(\Theta)}{\partial \boldsymbol{\theta}_\tau} = \frac{dL(\Theta)}{d\boldsymbol{\theta}}$$

Similarly, to see how the PES derivation works, consider a matrix of perturbations $\boldsymbol{\epsilon}$ and its vectorization $\text{vec}(\boldsymbol{\epsilon})$ as follows:

$$\boldsymbol{\epsilon} = \begin{bmatrix} \text{——} \boldsymbol{\epsilon}_1 \text{——} \\ \text{——} \boldsymbol{\epsilon}_2 \text{——} \\ \text{——} \boldsymbol{\epsilon}_3 \text{——} \end{bmatrix} = \begin{bmatrix} \epsilon_1^{(1)} & \epsilon_1^{(2)} \\ \epsilon_2^{(1)} & \epsilon_2^{(2)} \\ \epsilon_3^{(1)} & \epsilon_3^{(2)} \end{bmatrix} \qquad \text{vec}(\boldsymbol{\epsilon}) = \begin{bmatrix} \epsilon_1^{(1)} \\ \epsilon_2^{(1)} \\ \epsilon_3^{(1)} \\ \epsilon_1^{(2)} \\ \epsilon_2^{(2)} \\ \epsilon_3^{(2)} \end{bmatrix}$$

Then,

$$(\mathbf{I} \otimes \mathbf{1}^\top)\text{vec}(\boldsymbol{\epsilon}) = \begin{bmatrix} \epsilon_1^{(1)} + \epsilon_2^{(1)} + \epsilon_3^{(1)} \\ \epsilon_1^{(2)} + \epsilon_2^{(2)} + \epsilon_3^{(2)} \end{bmatrix} = \underbrace{\begin{bmatrix} \epsilon_1^{(1)} \\ \epsilon_1^{(2)} \end{bmatrix}}_{\boldsymbol{\epsilon}_1} + \underbrace{\begin{bmatrix} \epsilon_2^{(1)} \\ \epsilon_2^{(2)} \end{bmatrix}}_{\boldsymbol{\epsilon}_2} + \underbrace{\begin{bmatrix} \epsilon_3^{(1)} \\ \epsilon_3^{(2)} \end{bmatrix}}_{\boldsymbol{\epsilon}_3} = \sum_{\tau=1}^{T} \boldsymbol{\epsilon}_\tau$$

This shows how the following statements are equivalent in our derivation:

$$\frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}} \left[\left(\mathbf{I} \otimes \mathbf{1}^\top \right) \text{vec}\left(\boldsymbol{\epsilon} \right) L\left(\Theta + \boldsymbol{\epsilon} \right) \right] = \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}} \left[\left(\sum_{\tau=1}^{T} \boldsymbol{\epsilon}_\tau \right) L\left(\Theta + \boldsymbol{\epsilon} \right) \right]$$

F. Proof that PES is Unbiased

Statement F.1. *Let $\boldsymbol{\theta} \in \mathbb{R}^n$ and $L(\boldsymbol{\theta}) = \sum_{t=1}^{T} L_t(\boldsymbol{\theta})$. Suppose that $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ exists, and assume that L is quadratic, so that it is equivalent to its second-order Taylor series expansion:*

$$L(\Theta + \boldsymbol{\epsilon}) = L(\Theta) + \text{vec}(\boldsymbol{\epsilon})^\top \nabla_{\text{vec}(\Theta)} L(\Theta) + \frac{1}{2} \text{vec}(\boldsymbol{\epsilon})^\top \nabla^2_{\text{vec}(\Theta)} L(\Theta) \, \text{vec}(\boldsymbol{\epsilon})$$

Consider the PES estimator (using antithetic sampling) below:

$$\hat{\boldsymbol{g}}^{\textit{PES-A}} = (\mathbf{I} \otimes \mathbf{1}^\top) \mathbb{E}_{\boldsymbol{\epsilon}} \left[\frac{1}{2\sigma^2} \text{vec}(\boldsymbol{\epsilon}) (L(\Theta + \boldsymbol{\epsilon}) - L(\Theta - \boldsymbol{\epsilon})) \right],$$

where $\epsilon \sim \mathcal{N}(0, I\sigma^2)$.

Then, $\text{bias}(\hat{\boldsymbol{g}}^{\textit{PES-A}}) = \mathbb{E}_{\boldsymbol{\epsilon}}[\hat{\boldsymbol{g}}^{\textit{PES-A}}] - \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \mathbf{0}$.

Proof. Using the assumption that L is quadratic and due to antithetic sampling, we can simplify this expression $L(\Theta + \boldsymbol{\epsilon}) - L(\Theta - \boldsymbol{\epsilon})$ as follows:

$$\begin{aligned} L(\Theta + \boldsymbol{\epsilon}) - L(\Theta - \boldsymbol{\epsilon}) =& \cancel{(L(\Theta)} + \text{vec}(\boldsymbol{\epsilon})^\top \nabla_{\text{vec}(\Theta)} L(\Theta) + \cancel{\frac{1}{2} \text{vec}(\boldsymbol{\epsilon})^\top \nabla^2_{\text{vec}(\Theta)} L(\Theta) \, \text{vec}(\boldsymbol{\epsilon}))} && (22) \\ & - \cancel{(L(\Theta)} - \text{vec}(\boldsymbol{\epsilon})^\top \nabla_{\text{vec}(\Theta)} L(\Theta) + \cancel{\frac{1}{2} \text{vec}(\boldsymbol{\epsilon})^\top \nabla^2_{\text{vec}(\Theta)} L(\Theta) \, \text{vec}(\boldsymbol{\epsilon}))} && (23) \\ =& 2 \, \text{vec}(\boldsymbol{\epsilon})^\top \nabla_{\text{vec}(\Theta)} L(\Theta) && (24) \end{aligned}$$

Thus, we have:

$$\hat{\boldsymbol{g}}^{\text{PES-A}} = (\mathbf{I} \otimes \mathbf{1}^\top)\mathbb{E}_{\boldsymbol{\epsilon}}\left[\frac{1}{2\sigma^2} 2\,\text{vec}(\boldsymbol{\epsilon})\,\text{vec}(\boldsymbol{\epsilon})^\top \nabla_{\text{vec}(\Theta)} L(\Theta)\right] \tag{25}$$

$$= (\mathbf{I} \otimes \mathbf{1}^\top)\frac{1}{\sigma^2}\underbrace{\mathbb{E}_{\boldsymbol{\epsilon}}\left[\text{vec}(\boldsymbol{\epsilon})\,\text{vec}(\boldsymbol{\epsilon})^\top\right]}_{\sigma^2 I} \nabla_{\text{vec}(\Theta)} L(\Theta) \tag{26}$$

$$= (\mathbf{I} \otimes \mathbf{1}^\top)\nabla_{\text{vec}(\Theta)} L(\Theta) \tag{27}$$

$$= \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) \tag{28}$$

Thus, $\mathbb{E}[\hat{\boldsymbol{g}}^{\text{PES-A}}] = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$ and $\text{bias}(\hat{\boldsymbol{g}}^{\text{PES-A}}) = \mathbf{0}$. □

G. PES Variance

In this section, we derive the variance of PES. The PES estimator is defined as follows (we show the simplified form assuming antithetic sampling and quadratic functions L_t). To simplify the derivation here, we assume that the number of antithetic pairs is $N = 1$. As shorthand in the derivation, we write $\nabla L_t(\boldsymbol{\theta}) = \boldsymbol{g}_t$.

$$\hat{\boldsymbol{g}}^{\text{PES}} = \frac{1}{\sigma^2}\sum_{t=1}^{T} \boldsymbol{\xi}_t \boldsymbol{\epsilon}_t^\top \nabla L_t(\boldsymbol{\theta}) \tag{29}$$

$$= \frac{1}{\sigma^2}\left(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1 + \boldsymbol{\xi}_2\boldsymbol{\epsilon}_2^\top \boldsymbol{g}_2 + \cdots + \boldsymbol{\xi}_T\boldsymbol{\epsilon}_T^\top \boldsymbol{g}_T\right) \tag{30}$$

We use the total variance $\text{tr}(\text{Var}(\hat{\boldsymbol{g}}))$ to quantify the variance of the estimator:

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}})) = \text{tr}(\mathbb{E}[\hat{\boldsymbol{g}}\hat{\boldsymbol{g}}^\top] - \mathbb{E}[\hat{\boldsymbol{g}}]\mathbb{E}[\hat{\boldsymbol{g}}]^\top) \tag{31}$$

$$= \underbrace{\mathbb{E}[\hat{\boldsymbol{g}}^\top \hat{\boldsymbol{g}}]}_{①} - \underbrace{\mathbb{E}[\hat{\boldsymbol{g}}]^\top \mathbb{E}[\hat{\boldsymbol{g}}]}_{②} \tag{32}$$

Term ② is easy to compute, because we know that our estimator is unbiased, so $\mathbb{E}[\hat{\boldsymbol{g}}] = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$. Thus,

$$② = \mathbb{E}[\hat{\boldsymbol{g}}]^\top \mathbb{E}[\hat{\boldsymbol{g}}] = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})^\top \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = ||\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta})||_2^2 \tag{33}$$

To compute term ①, which is $\mathbb{E}[\hat{\boldsymbol{g}}^\top \hat{\boldsymbol{g}}]$, we will first expand $\hat{\boldsymbol{g}}^\top \hat{\boldsymbol{g}}$:

$$\hat{g}^\top \hat{g} = \frac{1}{\sigma^4}(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1 + \boldsymbol{\xi}_2\boldsymbol{\epsilon}_2^\top \boldsymbol{g}_2 + \cdots + \boldsymbol{\xi}_T\boldsymbol{\epsilon}_T^\top \boldsymbol{g}_T)^\top(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1 + \boldsymbol{\xi}_2\boldsymbol{\epsilon}_2^\top \boldsymbol{g}_2 + \cdots + \boldsymbol{\xi}_T\boldsymbol{\epsilon}_T^\top \boldsymbol{g}_T) \tag{34}$$

$$= \frac{1}{\sigma^4}[\underbrace{(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top g_1)^\top(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1)}_{ⓐ} + \underbrace{(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1)^\top(\boldsymbol{\xi}_2\boldsymbol{\epsilon}_2^\top \boldsymbol{g}_2)}_{ⓑ} + \cdots + (\boldsymbol{\xi}_T\boldsymbol{\epsilon}_T^\top \boldsymbol{g}_T)^\top(\boldsymbol{\xi}_T\boldsymbol{\epsilon}_T^\top \boldsymbol{g}_T)] \tag{35}$$

We have two types of terms appearing in the expression above:

- ⓐ **Terms where** $i = j$**:** $(\boldsymbol{\xi}_i\boldsymbol{\epsilon}_i^\top \boldsymbol{g}_i)^\top(\boldsymbol{\xi}_i\boldsymbol{\epsilon}_i^\top \boldsymbol{g}_i) = \boldsymbol{g}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\xi}_i^\top \boldsymbol{\xi}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{g}_i$
- ⓑ **Cross-terms (**$i \neq j$**) of the form:** $(\boldsymbol{\xi}_i\boldsymbol{\epsilon}_i^\top \boldsymbol{g}_i)^\top(\boldsymbol{\xi}_j\boldsymbol{\epsilon}_j^\top \boldsymbol{g}_j) = \boldsymbol{g}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\xi}_i^\top \boldsymbol{\xi}_j \boldsymbol{\epsilon}_j^\top \boldsymbol{g}_j$

Looking at terms of type ⓐ, we can first expand $\boldsymbol{\xi}_i^\top \boldsymbol{\xi}_i$ as follows:

$$\boldsymbol{\xi}_i^\top \boldsymbol{\xi}_i = (\boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_i)^\top(\boldsymbol{\epsilon}_1 + \cdots + \boldsymbol{\epsilon}_i) \tag{36}$$

$$= \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_2^\top \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_i + \cdots + \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i \tag{37}$$

Then, plugging in the expanded form of $\boldsymbol{\xi}_i^\top \boldsymbol{\xi}_i$ and taking the expectation, we have:

$$\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\xi}_i^\top \boldsymbol{\xi}_i \boldsymbol{\epsilon}_i^\top] = \mathbb{E}[\boldsymbol{\epsilon}_i(\boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_i + \cdots + \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i)\boldsymbol{\epsilon}_i^\top] \tag{38}$$

$$= \mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_i^\top + \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 \boldsymbol{\epsilon}_i^\top + \cdots + \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top + \cdots + \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top] \tag{39}$$

$$= \underbrace{\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_i^\top]}_{P\sigma^2 I} + \underbrace{\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 \boldsymbol{\epsilon}_i^\top]}_{0} + \cdots + \underbrace{\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top]}_{0} + \cdots + \underbrace{\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top]}_{(P+2)\sigma^4 I} \tag{40}$$

Terms like $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_i^\top]$ have the following form:

$$\mathbb{E}_{a,b}\left[ab^\top ba^\top\right] = \mathbb{E}_a[\mathbb{E}_b[ab^\top ba^\top]] = \mathbb{E}_a[a \underbrace{\mathbb{E}_b[b^\top b]}_{P\sigma^2} a^\top] = n\sigma^2 \underbrace{\mathbb{E}_a[aa^\top]}_{\sigma^2 I} = P\sigma^4 I \tag{41}$$

Where we obtained $\mathbb{E}_b[b^\top b] = P\sigma^2$ via:

$$\mathbb{E}_b[b^\top b] = \mathbb{E}_b[\text{tr}(b^\top b)] = \mathbb{E}_b[\text{tr}(bb^\top)] = \text{tr}\left(\mathbb{E}_b[bb^\top]\right) = \text{tr}(\sigma^2 I) = P\sigma^2 \tag{42}$$

Terms like $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 \boldsymbol{\epsilon}_i^\top]$ have the following form:

$$\mathbb{E}_{abc}[ab^\top ca^\top] = \mathbb{E}_a[\mathbb{E}_b[\mathbb{E}_c[ab^\top ca^\top]]] \tag{43}$$

$$= \mathbb{E}_a[\mathbb{E}_b[ab^\top \underbrace{\mathbb{E}_c[c]}_{\mu=0} a^\top]] \tag{44}$$

$$= 0 \tag{45}$$

Finally, for the single term where all the $\boldsymbol{\epsilon}$'s are equal, $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top]$, we used the following identity, which was derived in Appendix A.2 of (Maheswaranathan et al., 2018). The $\boldsymbol{\epsilon}_i$ are assumed to be drawn from $\mathcal{N}(\mathbf{0}, \Sigma)$, where in our case $\Sigma = \sigma^2 I$:

$$\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top] = \text{tr}(\Sigma)\Sigma + 2\Sigma^2 \tag{46}$$

$$= \text{tr}(\sigma^2 I)(\sigma^2 I) + 2(\sigma^2 I)^2 \tag{47}$$

$$= P\sigma^4 I + 2\sigma^4 I \tag{48}$$

$$= (P+2)\sigma^4 I \tag{49}$$

Now we need to figure out how many of each type of terms we have. There are two types of terms we are interested in (that are not equal to 0): 1) ones with repeated middle terms like: $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_i^\top]$ and $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_2^\top \boldsymbol{\epsilon}_2 \boldsymbol{\epsilon}_i^\top]$; and 2) ones with all terms equal like $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top]$ Any terms that have "mixed" middle terms like $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 \boldsymbol{\epsilon}_i^\top]$ will disappear. For a given i, we have one term $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top]$ and $i-1$ terms $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_i^\top]$. Thus, we have:

$$\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\xi}_i^\top \boldsymbol{\xi}_i \boldsymbol{\epsilon}_i^\top] = (i-1)P\sigma^4 I + (P+2)\sigma^4 I = [(i-1)P\sigma^4 + (P+2)\sigma^4]I$$

What about terms of type ⓑ, e.g. cross-terms: $\mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\xi}_i^\top \boldsymbol{\xi}_j \boldsymbol{\epsilon}_j^\top]$ where $i \neq j$? First, we can expand $\boldsymbol{\xi}_i^\top \boldsymbol{\xi}_j$ as follows:

$$\boldsymbol{\xi}_i^\top \boldsymbol{\xi}_j = (\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_i)^\top (\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_j) \tag{50}$$

$$= \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_j + \cdots + \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_j \tag{51}$$

Plugging in this expansion into the expectation, we have:

$$\mathbb{E}[\boldsymbol{\epsilon}_i(\boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 + \cdots + \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_j + \cdots + \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_j)\boldsymbol{\epsilon}_j^\top] = \mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_1 \boldsymbol{\epsilon}_j] + \mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_1^\top \boldsymbol{\epsilon}_2 \boldsymbol{\epsilon}_j] + \cdots + \mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_j \boldsymbol{\epsilon}_j^\top] \tag{52}$$

$$= \mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top \boldsymbol{\epsilon}_j \boldsymbol{\epsilon}_j^\top] \tag{53}$$

$$= \mathbb{E}[\boldsymbol{\epsilon}_i \boldsymbol{\epsilon}_i^\top]\mathbb{E}[\boldsymbol{\epsilon}_j \boldsymbol{\epsilon}_j^\top] \tag{54}$$

$$= (\sigma^2 I)(\sigma^2 I) \tag{55}$$

$$= \sigma^4 I \tag{56}$$

Putting this together, we have:

$$\mathbb{E}[\hat{\boldsymbol{g}}^\top \hat{\boldsymbol{g}}] = \frac{1}{\sigma^4}\mathbb{E}[(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1)^\top(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1)] + \mathbb{E}[(\boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top \boldsymbol{g}_1)^\top(\boldsymbol{\xi}_2\boldsymbol{\epsilon}_2^\top \boldsymbol{g}_2)] + \cdots + \mathbb{E}[(\boldsymbol{\xi}_T\boldsymbol{\epsilon}_T^\top \boldsymbol{g}_T)^\top(\boldsymbol{\xi}_T\boldsymbol{\epsilon}_T^\top \boldsymbol{g}_T)] \tag{57}$$

$$= \frac{1}{\sigma^4}\left(\boldsymbol{g}_1^\top \mathbb{E}[\boldsymbol{\epsilon}_1\boldsymbol{\xi}_1^\top \boldsymbol{\xi}_1\boldsymbol{\epsilon}_1^\top]\boldsymbol{g}_1 + \boldsymbol{g}_1^\top \mathbb{E}[\boldsymbol{\epsilon}_1\boldsymbol{\xi}_1^\top \boldsymbol{\xi}_2\boldsymbol{\epsilon}_2^\top]\boldsymbol{g}_2 + \cdots\right) \tag{58}$$

$$= \frac{1}{\sigma^4}\left(\sum_{t=1}^{T} \boldsymbol{g}_t^\top[((t-1)P\sigma^4 + (P+2)\sigma^4)I]\boldsymbol{g}_t + \sigma^4 \sum_{i\neq j} \boldsymbol{g}_i^\top \boldsymbol{g}_j\right) \tag{59}$$

$$= \sum_{t=1}^{T} \boldsymbol{g}_t^\top[((t-1)P + (P+2))I]\boldsymbol{g}_t + \sum_{i\neq j} \boldsymbol{g}_i^\top \boldsymbol{g}_j \tag{60}$$

$$= \sum_{t=1}^{T} ||\boldsymbol{g}_t||^2(Pt+2) + \sum_{i\neq j} \boldsymbol{g}_i^\top \boldsymbol{g}_j \tag{61}$$

To compute the total variance, we have:

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}})) = \mathbb{E}[\hat{\boldsymbol{g}}^\top \hat{\boldsymbol{g}}] - \mathbb{E}[\hat{\boldsymbol{g}}]^\top \mathbb{E}[\hat{\boldsymbol{g}}] = \sum_{t=1}^{T} ||\boldsymbol{g}_t||^2(Pt+2) + \cancel{\sum_{i\neq j} \boldsymbol{g}_i^\top \boldsymbol{g}_j} - \sum_{t=1}^{T} \boldsymbol{g}_t^\top \boldsymbol{g}_t - \cancel{\sum_{i\neq j} \boldsymbol{g}_i^\top \boldsymbol{g}_j} \tag{62}$$

$$= \sum_{t=1}^{T} ||\boldsymbol{g}_t||^2(Pt+2) - \sum_{t=1}^{T} ||\boldsymbol{g}_t||^2 \tag{63}$$

$$= \boxed{\sum_{t=1}^{T} ||\boldsymbol{g}_t||^2(Pt+1)} \tag{64}$$

G.1. Considering the dependence on T

Now we consider the dependence of the variance on the total number of unrolls T. Here, we denote by $\boldsymbol{g}$ the gradient of L, that is $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = \sum_{t=1}^{T} \nabla_{\boldsymbol{\theta}} L_t(\boldsymbol{\theta}) = \sum_{t=1}^{T} \boldsymbol{g}_t$. Consider two likely scenarios. For the first scenario, assume that the gradients for each unroll are i.i.d. In that case:

$$\mathbb{E}\left[||\boldsymbol{g}_t||^2\right] = \frac{1}{T}\mathbb{E}\left[||\boldsymbol{g}||^2\right] \tag{65}$$

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}})) = \frac{||\boldsymbol{g}||^2}{T}\sum_{t=1}^{T}(Pt+1) \tag{66}$$

$$= \frac{||\boldsymbol{g}||^2}{T}\left(\frac{1}{2}PT(T+1) + T\right) \tag{67}$$

$$= ||\boldsymbol{g}||^2\left(\frac{1}{2}P(T+1) + 1\right) \tag{68}$$

$$= ||\boldsymbol{g}||^2\left(\frac{1}{2}PT + \frac{1}{2}P + 1\right) \tag{69}$$

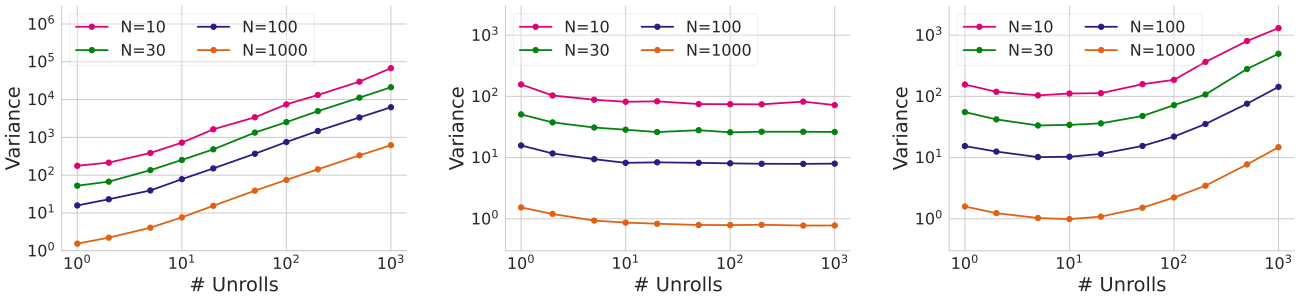
For the second scenario, assume that the gradients for each unroll are identical to each other. Then we have:

$$\boldsymbol{g}_t = \frac{1}{T}\boldsymbol{g} \tag{70}$$

$$||\boldsymbol{g}_t||^2 = \frac{1}{T^2}||\boldsymbol{g}||^2 \tag{71}$$

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}})) = ||\boldsymbol{g}||^2\left(\frac{1}{2}P + \frac{P}{2T} + \frac{1}{T}\right). \tag{72}$$

Figure 16 shows the empirical variance for several potential scenarios. We performed an analysis similar to that in Section 4, measuring the variance of the PES gradient with respect to the number of unrolls for a small LSTM on the Penn TreeBank (PTB) dataset. We constructed synthetic data sequences to illustrate different scenarios: in Figure 16a we used a 10^3 length sequence consisting of characters sampled uniformly at random from the PTB vocabulary, simulating the first scenario; in Figure 16b we used a 10^3 length sequence consisting of a single repeated character, simulating the second scenario; Figure 16c shows the variance for real data—the first 10^3 characters of PTB—which exhibits characteristics of both synthetic scenarios.



(a) Random sequence (b) Single character repeated (c) Real PTB sequence

Figure 16. **Empirical variance measurements for three scenarios.**

H. Reducing Variance by Incorporating the Analytic Gradient

For functions L that are differentiable, we can use the analytic gradient from the most recent partial unroll (e.g., backpropagating through the last K-step unroll) to reduce the variance of the PES gradient estimates. Below, we show how we can incorporate the analytic gradient in the ES estimate for $\frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}}$:

$$\frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}} \approx \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau \le t} \boldsymbol{\epsilon}_\tau\right) L_t(\Theta + \boldsymbol{\epsilon})\right] \tag{73}$$

$$= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau < t} \boldsymbol{\epsilon}_\tau\right) L_t(\Theta + \boldsymbol{\epsilon})\right] + \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}_t L_t(\Theta + \boldsymbol{\epsilon})] \tag{74}$$

$$= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau < t} \boldsymbol{\epsilon}_\tau\right) L_t(\Theta + \boldsymbol{\epsilon})\right] + \underbrace{\frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}_t}}_{\equiv \boldsymbol{p}_t} \tag{75}$$

$$= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau < t} \boldsymbol{\epsilon}_\tau\right) L_t(\Theta + \boldsymbol{\epsilon})\right] + \boldsymbol{p}_t - \underbrace{\frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau < t} \boldsymbol{\epsilon}_\tau\right) \boldsymbol{\epsilon}_t^\top \boldsymbol{p}_t\right]}_{=0} \tag{76}$$

$$= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau < t} \boldsymbol{\epsilon}_\tau\right) (L_t(\Theta + \boldsymbol{\epsilon}) - \boldsymbol{\epsilon}_t^\top \boldsymbol{p}_t)\right] + \boldsymbol{p}_t \tag{77}$$

We call the resulting estimator PES+Analytic. Algorithm 4 describes the implementation of this estimator, which requires a few simple changes from the standard PES estimator. We repeated the empirical variance measurement described in Section 4 and Appendix G.1 using the PES+Analytic estimator, for each of the three scenarios from Appendix G.1, shown in Figure 18. Similarly to the other variance measurements, we report variance normalized by the squared norm of the true gradient. We found that variance increases with the number of unrolls, but the PES+Analytic variance is 1-2 orders of magnitude smaller than the standard PES variance.

Algorithm 3 Original persistent evolution strategies (PES) estimator, identical to Section 4.

Input: s_0, initial state
K, truncation length for partial unrolls
N, number of particles
σ, standard deviation of perturbations
α, learning rate for PES optimization

Initialize $\boldsymbol{s}^{(i)} = \boldsymbol{s}_0$ for $i \in \{1, \dots, N\}$
Initialize $\boldsymbol{\xi}^{(i)} \leftarrow \mathbf{0}$ for $i \in \{1, \dots, N\}$
repeat

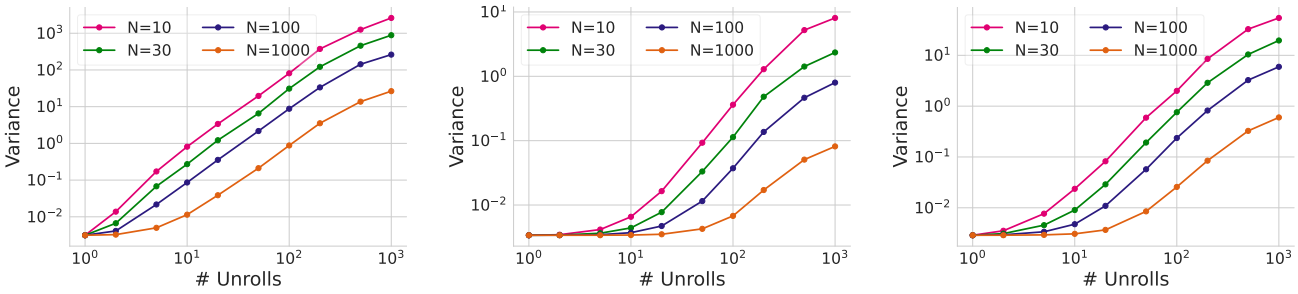
$\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \mathbf{0}$
for $i = 1, \dots, N$ **do**
$\boldsymbol{\epsilon}^{(i)} \leftarrow \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 I) & i \text{ odd} \\ -\boldsymbol{\epsilon}^{(i-1)} & i \text{ even} \end{cases}$
$\boldsymbol{s}^{(i)}, \hat{L}_K^{(i)} \leftarrow \text{unroll}(\boldsymbol{s}^{(i)}, \boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}, K)$
$\boldsymbol{\xi}^{(i)} \leftarrow \boldsymbol{\xi}^{(i)} + \boldsymbol{\epsilon}^{(i)}$
$\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \hat{\boldsymbol{g}}^{\text{PES}} + \boldsymbol{\xi}^{(i)} \hat{L}_K^{(i)}$
end for
$\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{PES}}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{PES}}$

Algorithm 4 PES + analytic gradient. Differences from PES are highlighted in purple.

Input: s_0, initial state
K, truncation length for partial unrolls
N, number of particles
σ, standard deviation of perturbations
α, learning rate for PES optimization

Initialize $\boldsymbol{s} = \boldsymbol{s}_0$
Initialize $\boldsymbol{s}^{(i)} = \boldsymbol{s}_0$ for $i \in \{1, \dots, N\}$
Initialize $\boldsymbol{\xi}^{(i)} \leftarrow \mathbf{0}$ for $i \in \{1, \dots, N\}$
repeat
$\boldsymbol{s}, L \leftarrow \text{unroll}(\boldsymbol{s}, \boldsymbol{\theta}, K)$
$\boldsymbol{p} \leftarrow \nabla_{\boldsymbol{\theta}} L$
$\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \mathbf{0}$
for $i = 1, \dots, N$ **do**
$\epsilon^{(i)} \leftarrow \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 I) & i \text{ odd} \\ -\boldsymbol{\epsilon}^{(i-1)} & i \text{ even} \end{cases}$
$\boldsymbol{s}^{(i)}, \hat{L}_K^{(i)} \leftarrow \text{unroll}(\boldsymbol{s}^{(i)}, \boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}, K)$
$\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \hat{\boldsymbol{g}}^{\text{PES}} + \boldsymbol{\xi}^{(i)} (\hat{L}_K^{(i)} - \boldsymbol{\epsilon}^{(i)\top} \boldsymbol{p})$
$\boldsymbol{\xi}^{(i)} \leftarrow \boldsymbol{\xi}^{(i)} + \boldsymbol{\epsilon}^{(i)}$
end for
$\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{PES}} + \boldsymbol{p}$
$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{PES}}$

Figure 17. **A comparison of the PES and PES+Analytic gradient estimators**, applied to partial unrolls of a computation graph. The conditional statement for $\boldsymbol{\epsilon}^{(i)}$ is used to implement antithetic sampling. For clarity, we describe the meta-optimization updates to $\boldsymbol{\theta}$ using SGD, but we typically use Adam in practice.



(a) Random sequence (b) Single character repeated (c) Real PTB sequence

Figure 18. **Empirical variance measurements for three scenarios, incorporating the analytic gradient from the most recent unroll to reduce variance.**

I. Connection to Gradient Estimation in Stochastic Computation Graphs

In this section, we show how PES can be derived using the framework for gradient estimation in stochastic computation graphs introduced in (Schulman et al., 2015). We follow their notation for this exposition: in Figure 19, squares represent deterministic nodes, which are functions of their parents; circles represent stochastic nodes which are distributed conditionally on their parents, and nodes not in squares or circles represent inputs. For notational simplicity, in the following exposition we consider 1-dimensional θ. We represent the unrolled computation graph in terms of an input node θ, that gives rise to a stochastic variable θ_t at each time step; the sampled θ_t is used to compute the state s_t, which is a deterministic function of the previous state s_{t-1} and the current parameters θ_t. The losses L_t are designated as cost nodes, and our objective is $L = \sum_t L_t$.

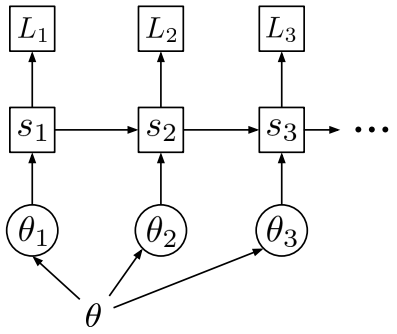


Figure 19. **Unrolled stochastic computation graph for PES**, in the notation of (Schulman et al., 2015).

Theorem 1 from (Schulman et al., 2015) gives the following general form for the gradient of the sum of cost nodes in such a stochastic computation graph. Here, $\mathcal{C}$ is the set of cost nodes; $\mathcal{S}$ is the set of stochastic nodes; DEPS_w denotes the set of nodes that w *depends on*; $a \prec^D b$ indicates that node a node depends deterministically on node b (note that this relationship holds as long as there are no stochastic nodes along a path from a to b; in our case, $\theta \prec^D \theta_t$ holds for all t); and $\hat{Q}_w$ is the sum of cost nodes downstream from node w.

$$\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{c\in\mathcal{C}} c\right] = \mathbb{E}\left[\sum_{w\in\mathcal{S},\theta\prec^D w}\left(\frac{\partial}{\partial\theta}\log p(w|\text{DEPS}_w)\right)\hat{Q}_w + \sum_{c\in\mathcal{C},\theta\prec^D c}\frac{\partial}{\partial\theta}c(\text{DEPS}_c)\right] \tag{78}$$

For the computation graph in Figure 19, θ does not deterministically influence any of the cost nodes L_t, so the second term in the expectation in Eq. 78 will be 0. In addition, each stochastic node θ_t, depends only on θ, e.g. $\text{DEPS}_{\theta_t} = \{\theta\}, \forall t$. Thus, our gradient estimate is:

$$\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{t=1}^{T} L_t\right] = \mathbb{E}\left[\sum_{t=1}^{T}\left(\frac{\partial}{\partial\theta}\log p(\theta_t|\theta)\right)\hat{Q}_{\theta_t}\right] \tag{79}$$

$\hat{Q}_{\theta_t}$ is the sum of cost nodes downstream of θ_t, thus $\hat{Q}_{\theta_t} = \sum_{i=t}^{T} L_i$. Now, each $\theta_t \sim \mathcal{N}(\theta, \sigma^2)$, so we have:

$$\log p(\theta_t \mid \theta) = \log\mathcal{N}(\theta_t \mid \theta, \sigma^2) = \log\frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{2\sigma^2}(\theta_t - \theta)^2 \tag{80}$$

Then,

$$\begin{aligned}
\frac{\partial}{\partial\theta}\log p(\theta_t \mid \theta) &= -\frac{1}{2\sigma^2}\cdot 2(\theta_t - \theta)\cdot(-1) && (81)\\
&= \frac{1}{\sigma^2}(\theta_t - \theta) && (82)\\
&= \frac{1}{\sigma^2}(\theta + \epsilon_t - \theta) && (83)\\
&= \frac{1}{\sigma^2}\epsilon_t && (84)
\end{aligned}$$

where we used the reparameterization $\theta_t = \theta + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$. Plugging this into Eq. 79, we have:

$$\begin{aligned}
\frac{\partial}{\partial\theta}\mathbb{E}\left[\sum_{t=1}^{T} L_t\right] &= \mathbb{E}\left[\sum_{t=1}^{T}\frac{1}{\sigma^2}\epsilon_t\hat{Q}_{\theta_t}\right] && (85)\\
&= \frac{1}{\sigma^2}\mathbb{E}\left[\epsilon_1(L_1 + L_2 + \cdots + L_T) + \epsilon_2(L_2 + L_3 + \cdots + L_T) + \cdots + \epsilon_T L_T\right] && (86)\\
&= \frac{1}{\sigma^2}\mathbb{E}[\epsilon_1 L_1 + (\epsilon_1 + \epsilon_2)L_2 + (\epsilon_1 + \epsilon_2 + \epsilon_3)L_3 + \cdots + (\epsilon_1 + \cdots + \epsilon_T)L_T] && (87)\\
&= \frac{1}{\sigma^2}\mathbb{E}\left[\sum_{t=1}^{T}\left(\sum_{\tau=1}^{t}\epsilon_\tau\right)L_t\right] && (88)
\end{aligned}$$

Eq. 88 recovers the PES estimator.

J. Derivations and Compute/Memory Costs

BPTT, TBPTT, ARTBP. Backpropagating through a full unroll of T steps requires T forward and backward passes, yielding compute $T(F+B)$; all T states must be stored in memory to be available for gradient computation during backprop, yielding memory cost TS. Similarly, because TBPTT unrolls the computation graph for K steps, it requires K forward and backward passes, yielding computation $K(F+B)$, and requires storing K states in memory, yielding memory cost KS. ARTBP is identical to TBPTT except that it randomly samples the truncation length in a theoretically-justified way to reduce or eliminate truncation bias. In theory, the sampled truncation lengths must allow for maximum length T, yielding worst-case compute $T(F+B)$ and memory cost TS. However, in practice this is often intractable, so truncation lengths may be sampled within a restricted range centered around K—this is no longer unbiased, but yields average case compute $K(F+B)$ and memory cost KS (which is reported in Table 1).

RTRL. We begin by deriving RTRL, which simply corresponds to forward-mode differentiation. Let the state be $\boldsymbol{s}_t \in \mathbb{R}^S$ and the parameters be $\boldsymbol{\theta} \in \mathbb{R}^P$. We have a dynamical system defined by:

$$\boldsymbol{s}_t = f(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t, \boldsymbol{\theta}) \tag{89}$$

and our objective is $L = \sum_{t=1}^{T} L_t$. In order to optimize this objective, we need the gradient $\nabla_{\boldsymbol{\theta}} L = \sum_{t=1}^{T} \frac{dL_t}{d\boldsymbol{\theta}}$. The loss at step t is a function of $\boldsymbol{s}_t$, so we have:

$$\frac{dL_t(\boldsymbol{s}_t)}{d\boldsymbol{\theta}} = \frac{\partial L_t}{\partial \boldsymbol{s}_t} \frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}} \tag{90}$$

Using Eq. 89 and the chain rule, we have:

$$\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}} = \frac{df(\boldsymbol{s}_{t-1}, \boldsymbol{x}_t, \boldsymbol{\theta})}{d\boldsymbol{\theta}} \tag{91}$$

$$= \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_{t-1}} \frac{d\boldsymbol{s}_{t-1}}{d\boldsymbol{\theta}} + \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{x}_t} \cancelto{0}{\frac{d\boldsymbol{x}_t}{d\boldsymbol{\theta}}} + \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{\theta}} \cancelto{1}{\frac{d\boldsymbol{\theta}}{d\boldsymbol{\theta}}} \tag{92}$$

$$= \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_{t-1}} \frac{d\boldsymbol{s}_{t-1}}{d\boldsymbol{\theta}} + \frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{\theta}} \tag{93}$$

Thus, we have the recurrence relation:

$$\underbrace{\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}}}_{G_t} = \underbrace{\frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{s}_{t-1}}}_{H_t} \underbrace{\frac{d\boldsymbol{s}_{t-1}}{d\boldsymbol{\theta}}}_{G_{t-1}} + \underbrace{\frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{\theta}}}_{F_t} \tag{94}$$

Here, G_t is $S \times P$, H_t is $S \times S$, and F_t is $S \times P$. RTRL maintains the Jacobian G_t, which requires memory SP; furthermore, instantiating the matrices H_t and F_t requires memory S^2 and SP, respectively, so the total memory cost of RTRL is $2SP + S^2$. The matrix multiplication $H_t G_{t-1}$ has computational complexity $S^2 P$. The cost of computing the Jacobian F_t is approximately $\min\{S(F+B), P(F+B)\}$, depending on which of S or P is smaller-dimensional (and correspondingly whether we use forward-mode or reverse-mode automatic differentiation to compute the rows/columns of the Jacobian). Similarly, the cost of computing the Jacobian H_t is approximately $S(F+B)$ (using either forward or reverse mode autodiff). Thus, the total computational cost of RTRL is: $S^2 P + S(F+B) + \min\{S(F+B) + P(F+B)\}$.

Note that, in general, it matters which of $\boldsymbol{s}_t$ or $\boldsymbol{\theta}$ is higher dimensional. In the case of unrolled optimization, S is usually larger than P, causing RTRL to be particularly memory-intensive due to the $S \times S$ Jacobian H_t. The computation and memory costs we have derived here are expressed in a general form for state and parameter dimensions S and P, respectively. In the case of RNN training, most prior work (such as (Tallec & Ollivier, 2017a; Mujika et al., 2018; Benzing et al., 2019)) assumes that the RNN parameters are of dimensionality S^2, where S is the size of the hidden state. [5]

[5] This is a simplification of the parameter count for RNNs, assuming that it is dominated by the hidden-to-hidden weight matrix.

UORO. Unbiased Online Recurrent Optimization (UORO) (Tallec & Ollivier, 2017a) approximates RTRL by maintaining a rank-1 estimate of the Jacobian G_t as:

$$G_t \approx \tilde{\boldsymbol{s}}_t \tilde{\boldsymbol{\theta}}_t^\top \tag{95}$$

where $\tilde{\boldsymbol{s}}_t$ and $\tilde{\boldsymbol{\theta}}_t$ are vectors of dimensions S and P, respectively. Ultimately, we are interested in the gradient $\frac{\partial L_t}{\partial \boldsymbol{\theta}}$. Using the UORO approximation to G_t, we can write the gradient as follows:

$$\begin{aligned}
\frac{\partial L_t}{\partial \boldsymbol{\theta}} &= \frac{\partial L_t}{\partial \boldsymbol{s}_t}\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}} && (96)\\
&= \frac{\partial L_t}{\partial \boldsymbol{s}_t} G_t && (97)\\
&= \frac{\partial L_t}{\partial \boldsymbol{s}_t}(H_t G_{t-1} + F_t) && (98)\\
&= \frac{\partial L_t}{\partial \boldsymbol{s}_t}(H_t(\tilde{\boldsymbol{s}}_t\tilde{\boldsymbol{\theta}}_t^\top) + F_t) && (99)\\
&= \frac{\partial L_t}{\partial \boldsymbol{s}_t}(H_t(\tilde{\boldsymbol{s}}_t\tilde{\boldsymbol{\theta}}_t^\top)) + \frac{\partial L_t}{\partial \boldsymbol{s}_t}F_t && (100)\\
&= \frac{\partial L_t}{\partial \boldsymbol{s}_t}(H_t(\tilde{\boldsymbol{s}}_t\tilde{\boldsymbol{\theta}}_t^\top)) + \frac{\partial L_t}{\partial \boldsymbol{s}_t}\frac{\partial \boldsymbol{s}_t}{\partial \boldsymbol{\theta}} && (101)\\
&= \frac{\partial L_t}{\partial \boldsymbol{s}_t}(H_t(\tilde{\boldsymbol{s}}_t\tilde{\boldsymbol{\theta}}_t^\top)) + \frac{\partial L_t}{\partial \boldsymbol{\theta}} && (102)\\
&= \underbrace{\left(\frac{\partial L_t}{\partial \boldsymbol{s}_t}H_t\tilde{\boldsymbol{s}}_t\right)}_{1\times 1}\tilde{\boldsymbol{\theta}}_t^\top + \underbrace{\frac{\partial L_t}{\partial \boldsymbol{\theta}}}_{1\times P} && (103)
\end{aligned}$$

Here, $\frac{\partial L_t}{\partial \boldsymbol{s}_t}$ is $1 \times S$, H_t is $S \times S$, $\tilde{\boldsymbol{s}}_t$ is $S \times 1$, $\tilde{\boldsymbol{\theta}}_t$ is $1 \times P$, and F_t is $S \times P$.

This leads to a total computation cost of $F + B + S^2 + P$. We require one pass of backprop to compute the partial derivative, a vector-matrix product size S by $S \times S$ (S^2), then element-wise operations on the full parameter space (P). The memory cost of storing both $\tilde{\boldsymbol{s}}_t$ and $\tilde{\boldsymbol{\theta}}_t$ is $S + P$.

Reparameterization. The reparameterization gradient estimator is $\hat{g}^{\text{reparam}} = \frac{1}{N}\sum_{i=1}^{N} \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta} + \sigma\boldsymbol{\epsilon}^{(i)})$, where $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(0, I)$. With respect to computational complexity, this is equivalent to BPTT: its compute cost is $T(F + B)$ and its memory cost is TS.

ES. ES applied to an unroll of length K requires performing K forward passes—it does not require any backward passes, since ES is not gradient-based (e.g., it is a zeroth-order optimization algoritm). Because ES does not require backprop, it does not need to store the intermediate states in memory, only the most recent state, yielding memory cost S that is independent of the unroll length. Using ES with N particles yields total compute and memory costs NKF and NS, respectively.

PES. As PES is an evolutionary strategies-based method, it also does not require backward passes; applied to unrolls of length K, PES has compute cost KF. In addition to storing the current state of size S as in ES, PES also maintains a perturbation accumulator for each particle; thus, the memory cost of a single PES chain is $S + P$. Using PES with N particles yields total compute and memory costs NKF and $N(S + P)$, respectively.

PES+Analytic. Similarly to standard PES, we need to maintain a collection of N states, each of size S, and N perturbation accumulators, each of size P, yielding memory cost $N(S + P)$; unrolling each state for K steps requires computational cost NKF. To incorporate the analytic gradient, we need to maintain one additional particle that is unrolled using the mean $\boldsymbol{\theta}$ rather than a perturbed version $\boldsymbol{\theta} + \boldsymbol{\epsilon}$; this adds memory cost S. The main computational and memory overhead comes from the gradient computation through the partial unroll of length K: similarly to TBPTT, this requires storing K intermediate states, yielding memory cost KS, and requires K forward and K backward operations, yielding computational cost $K(F + B)$. Combined with the memory and computational cost of standard PES, we have total compute cost $NKF + K(F + B)$ and total memory cost $N(S + P) + (K + 1)S$.

K. Diagrammatic Representation of Algorithms

Figure 20 provides diagrammatic representations of ES and PES. For each partial unroll, vanilla ES starts from a shared initial state $\boldsymbol{s}^{(0)}$ that is evolved in parallel using perturbed parameters $\boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}$. After each truncated unroll, the mean parameters $\boldsymbol{\theta}$ are used to update the state, which then becomes the initial state for the next truncated unroll; no information is passed between truncated unrolls for vanilla ES. In contrast, PES maintains a set of states $\boldsymbol{s}^{(i)}$ that are evolved in parallel, each according to a different perturbation of the parameters $\boldsymbol{\theta}$ in each truncated unroll. Intuitively, these states maintain their history between truncated unrolls, since we accumulate the perturbations experienced by each state over the course of meta-optimization; when we reach the end of an inner problem, the states are reset to the same initialization, and the perturbation accumulators are reset to $\mathbf{0}$.

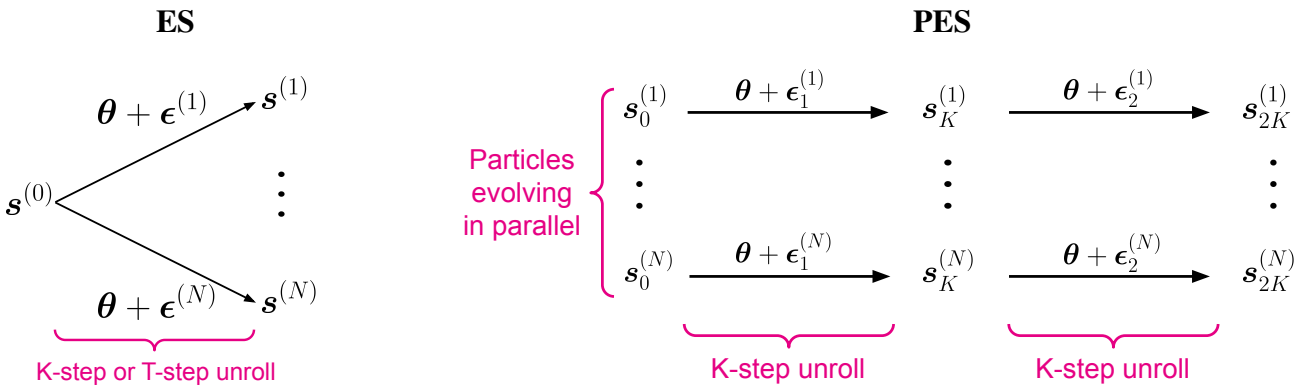


Figure 20. **Left:** Evolution strategies (ES). **Right:** Persistent evolution strategies (PES).

L. Ablation Studies

In this section, we show an ablation study over the the number of particles N, and the truncation length K (which controls the number of unrolls per inner-problem). In Figure 21 we show the sensitivity of PES to these meta-parameters for a version of the 2D regression problem (from Section 5.4) with total inner problem length $T = 10,000$.

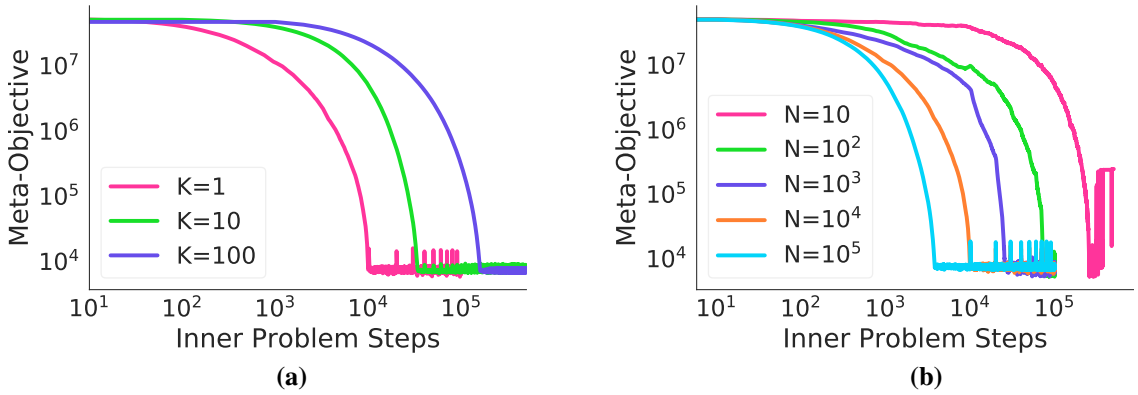


Figure 21. Ablation over meta-parameters for PES applied to the toy 2D regression task with total number of inner steps $T = 10,000$. Here we vary the truncation length K and number of particles N; all other meta-parameters are fixed: we used Adam with learning rate 3e-2 for meta-optimization, and perturbation standard deviation 0.1. **(a)** Decreasing K yields shorter truncations, which allow for more frequent meta-updates, improving performance compared to longer truncations. For these runs, $N = 10^4$. **(b)** As in standard ES, increasing the particle count for PES reduces variance and can yield substantial improvements in terms of inner iterations performed, or wall-clock time. For these runs, $K = 1$.

M. Implementation

Code Listing 1 presents a simple JAX implementation of the toy 2D regression meta-learning problem from Section 5.4, in a self-contained, runnable example. PES is easy to implement efficiently in JAX by making use of the construct `jax.vmap` (or `jax.pmap` in settings with multiple workers) to parallelize the unrolling computations over N particles.

Listing 1. Simplified PES implementation in JAX, for the 2D regression problem from Section 5.4.

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

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_argnums=(3,4))
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_argnums=(5,6,7,8))
def pes_grad(key, xs, pert_accum, theta, t0, T, K, sigma, N):
   """Compute PES gradient estimate.

   Args:
     key: JAX PRNG key
     xs: Nx2 array of particles/states to be updated
     pert_accum: Nx2 array of accumlated perturbations for each particle
     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: PES gradient estimate
      xs: Nx2 array of updates particles/states
      pert_accum: Nx2 array of updated perturbations for each particle
    """
    # Generate antithetic perturbations
    pos_perts = jax.random.normal(key, (N//2, theta.shape[0])) * sigma # Antithetic
        positives
    neg_perts = -pos_perts # Antithetic negatives
    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)
    # Add the perturbations from this unroll to the perturbation accumulators
    pert_accum = pert_accum + perts
    # Compute the PES gradient estimate
    theta_grad = jnp.mean(pert_accum * L.reshape(-1, 1) / (sigma**2), axis=0)
    return theta_grad, xs, pert_accum

opt_params = { 'lr': 1e-2, 'b1': 0.99, 'b2': 0.999, 'eps': 1e-8,
               'm': jnp.zeros(2),
               'v': jnp.zeros(2) }

def outer_optimizer_step(params, grads, opt_params, t):
    lr = opt_params['lr']
    b1 = opt_params['b1']
    b2 = opt_params['b2']
    eps = opt_params['eps']
    opt_params['m'] = (1 - b1) * grads + b1 * opt_params['m']
    opt_params['v'] = (1 - b2) * (grads**2) + b2 * opt_params['v']
    mhat = opt_params['m'] / (1 - b1**(t+1))
    vhat = opt_params['v'] / (1 - b2**(t+1))
    updated_params = params - lr * mhat / (jnp.sqrt(vhat) + eps)
    return updated_params, opt_params

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])
pert_accum = jnp.zeros((N, theta.shape[0]))

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

    theta_grad, xs, pert_accum = pes_grad(skey, xs, pert_accum, theta, t, T, K, sigma, N)
    theta, opt_params = outer_optimizer_step(theta, theta_grad, opt_params, i)
    t += K

    if i % 100 == 0:
        L, _ = unroll(jnp.array([1.0, 1.0]), theta, 0, T, T) # Run a full unroll to get the
            cost
```

```
        print(i, jnp.exp(theta), theta_grad, L)
```