Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence (IJCAI-22)
Sister Conferences Best Papers Track

Unbiased Gradient Estimation in Unrolled Computation Graphs with Persistent Evolution Strategies (Extended Abstract)*

Paul Vicol[1] , **Luke Metz**[2] , **Jascha Sohl-Dickstein**[2]
[1]University of Toronto
[2]Google Brain
pvicol@cs.toronto.edu, {lmetz, jaschasd}@google.com

Abstract

Current approaches for optimizing parameters in unrolled computation graphs suffer from high variance gradients, bias, slow updates, or large memory usage. We introduce a method called Persistent Evolution Strategies (PES), which divides the computation graph into a series of truncated unrolls, and performs an evolution strategies-based update step after each unroll. PES eliminates bias from these truncations by accumulating correction terms over the entire sequence of unrolls. PES allows for rapid parameter updates, has low memory usage, is unbiased, and has reasonable variance.

1 Introduction

Unrolled computation graphs arise in many scenarios in machine learning, including when training RNNs [Williams and Peng, 1990], tuning hyperparameters through unrolled computation graphs [Franceschi *et al.*, 2018], and training learned optimizers [Wichrowska *et al.*, 2017; Metz *et al.*, 2018].

Classic algorithms for computing gradients in such unrolled computation graphs include reverse- and forward-mode gradient accumulation. Backpropagation through time (BPTT) involves backpropagating through a full unrolled sequence for each parameter update. This faces several difficulties: 1) the memory cost scales linearly with the unroll length, due to storage of intermediate activations for backprop 2) it only performs one parameter update after each full unroll, which is expensive and introduces large latency between parameter updates; 3) long unrolls can lead to exploding or vanishing gradients [Pascanu *et al.*, 2013], and chaotic loss landscapes [Parmas *et al.*, 2018; Metz *et al.*, 2019]. A common technique to alleviate these issues is truncated backprop through time (TBPTT) [Werbos, 1990], which splits the full sequence into shorter subsequences and performs an update after processing each subsequence. However, TBPTT yields biased gradients that can severely impact training.

An alternative to BPTT is real-time recurrent learning (RTRL), which performs forward gradient accumulation [Williams and Zipser, 1989]. RTRL enables online parameter updates (after each partial unroll) and does not suffer from truncation bias; however, its memory and compute requirements render it intractable for large-scale problems. Many approximations to RTRL have been proposed [Tallec and Ollivier, 2017; Mujika *et al.*, 2018; Benzing *et al.*, 2019], but most have high variance, are complicated to implement, or are only applicable to a restricted class of models.

To address the poorly conditioned loss surfaces that often result from sequential computation, it can be useful to minimize a smoothed version of the loss. Evolution strategies (ES) is a family of algorithms that estimate gradients using stochastic finite-differences, and which provide an unbiased estimate of the gradient of the objective smoothed with a Gaussian. We introduce an approach to unbiased gradient estimation using short, truncated unrolls, called Persistent Evolution Strategies (PES). In PES, we accumulate the perturbations experienced by the *outer parameters* in each partial unroll—rather than starting perturbations from scratch as in ES—which yields an unbiased estimate of the gradient even when using truncated sequences. PES is simple to implement, and because it is ES-based, it retains desirable characteristics such as being parallelizable, memory efficient, and broadly applicable to many problems, including to non-differentiable target functions.

Problem Setup. We consider unrolled computation graphs with state $\boldsymbol{s}_t$ updated based on parameters $\boldsymbol{\theta}$ via the recurrence:

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

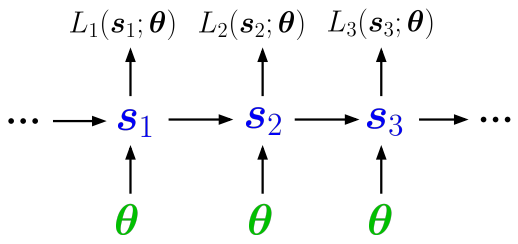
where $\boldsymbol{x}_t$ is an optional input at step t. The objective function for optimizing $\boldsymbol{\theta}$ is the sum of per-timestep losses $L_t(\boldsymbol{s}_t; \boldsymbol{\theta})$:

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

This setup is general, even encompassing situations where we want to consider only the final loss at step T, which can be expressed using a telescoping sum of loss differences between successive steps [Beatson and Adams, 2019]. Instances of this problem setup include training RNNs, training learned optimizers, learning policies for control tasks, and hyperparameter optimization, as illustrated in Figure 1.

Smoothing & Evolution Strategies (ES). Unrolling optimization for many steps can lead to pathological meta-loss

*This is an extended abstract of [Vicol *et al.*, 2021] that received the best paper award at ICML 2021.



Objective: $L(\boldsymbol{\theta}) = \sum_t L_t(\boldsymbol{s}_t;\boldsymbol{\theta})$

Task	$\boldsymbol{s}_t$	$\boldsymbol{\theta}$
RNN	Hidden State	RNN Parameters
Hyperparameter Optimization	Model Parameters	Hyperparameters
Learned Optimizers	Model Parameters	Learned Optimizer Parameters
RL	Environment State	Policy Parameters

Figure 1: **An unrolled computation graph**, illustrating how several tasks can be described using Equations 1 and 2. For example, in unrolled optimization, $\boldsymbol{s}_t$ contains the parameters of the base model and optimizer accumulators (e.g. momentum), $L_t(\cdot)$ is a meta-objective such as validation performance, $\boldsymbol{\theta}$ contains hyperparameters (e.g. the learning rate, weight decay, etc.) that govern the optimization, and f corresponds to the update step of an optimization algorithm such as SGD, RMSprop [Tieleman and Hinton, 2012], or Adam [Kingma and Ba, 2015].

surfaces that exhibit near-discontinuities and chaotic structure [Parmas *et al.*, 2018; Metz *et al.*, 2019]. Optimization on such non-smooth landscapes fails due to exploding gradients or gets stuck in poor local minima. An effective method to address these pathologies is to smooth the meta-loss surface, e.g. descend the Gaussian-blurred objective $L(\boldsymbol{\theta}) = \mathbb{E}_{\tilde{\boldsymbol{\theta}}\sim\mathcal{N}(\boldsymbol{\theta},\sigma^2 I)}[L(\tilde{\boldsymbol{\theta}})]$ [Staines and Barber, 2012; Metz *et al.*, 2019]. Evolution strategies provides an unbiased estimate of the gradient $\nabla_{\boldsymbol{\theta}}\mathbb{E}_{\tilde{\boldsymbol{\theta}}\sim\mathcal{N}(\boldsymbol{\theta},\sigma^2 I)}[L(\tilde{\boldsymbol{\theta}})]$. The ES estimator is defined as: $\hat{\boldsymbol{g}}^{\text{ES}} = \frac{1}{N\sigma^2}\sum_{i=1}^{N}\boldsymbol{\epsilon}^{(i)}L(\boldsymbol{\theta}+\boldsymbol{\epsilon}^{(i)})$, where $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(0,\sigma^2 I)$. ES is trivially parallelizable, and thus highly scalable; it has seen renewed interest as a viable algorithm for reinforcement learning among other black-box problems [Ha and Schmidhuber, 2018; Cui *et al.*, 2018; Ha, 2020]. However, applying ES to full unrolls is costly, while using partial unrolls leads to truncation bias, similarly to TBPTT.

2 Persistent Evolution Strategies

We introduce a method to obtain unbiased gradient estimates from partial unrolls of a computation graph, called Persistent Evolution Strategies (PES). We first derive the PES gradient estimator, and then present a practical algorithm (Algorithm 2).

Derivation. 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 $\boldsymbol{\theta}_t = \boldsymbol{\theta}, \forall t$. 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 $L_t(\Theta)$. 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 $\frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)} \in \mathbb{R}^{PT\times 1}$, and then using ES to approximate $\frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)}$,

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

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

where $\otimes$ denotes the Kronecker product, $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\epsilon}_T)^\top$ 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 i.i.d. 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,

$$\begin{aligned}\boldsymbol{g}^{\text{PES}} &= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau=1}^{T}\boldsymbol{\epsilon}_\tau\right)\sum_{t=1}^{T}L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right]\\ &= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\left(\sum_{\tau=1}^{t}\boldsymbol{\epsilon}_\tau\right)L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right] \qquad (3)\\ &= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}}\right], \qquad (4)\end{aligned}$$

$$\hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}} = \frac{1}{\sigma^2}\boldsymbol{\xi}_t L_t\left(\boldsymbol{\theta}_1+\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\theta}_t+\boldsymbol{\epsilon}_t\right). \qquad (5)$$

where $\boldsymbol{\xi}_t = \sum_{\tau=1}^{t}\boldsymbol{\epsilon}_\tau$, Equation 3 relies on $L_t(\cdot)$ being independent of $\boldsymbol{\epsilon}_\tau$ for $\tau > t$, and Equation 5 similarly relies on $L_t(\cdot)$ only being a function of $\boldsymbol{\theta}_\tau$ for $\tau \le t$. The PES estimator consists of Monte Carlo estimates of Equation 4,

$$\hat{\boldsymbol{g}}^{\text{PES}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}^{(i)}} \qquad (6)$$

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. When using antithetic sampling (e.g., positive and negative perturbation pairs in Eq. 6), we denote the estimator by $\hat{\boldsymbol{g}}^{\text{PES-A}}$. See the full paper for a proof of the following Statement 2.1.

Statement 2.1 (PES is unbiased)**.** *Let $\boldsymbol{\theta} \in \mathbb{R}^P$ 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: $L(\Theta+\boldsymbol{\epsilon}) = L(\Theta)+\operatorname{vec}(\boldsymbol{\epsilon})^\top\nabla_{\operatorname{vec}(\Theta)}L(\Theta)+\frac{1}{2}\operatorname{vec}(\boldsymbol{\epsilon})^\top\nabla^2_{\operatorname{vec}(\Theta)}L(\Theta)\operatorname{vec}(\boldsymbol{\epsilon})$. Then, $\operatorname{bias}(\hat{\boldsymbol{g}}^{\text{PES-A}}) = \mathbb{E}_{\boldsymbol{\epsilon}}[\hat{\boldsymbol{g}}^{\text{PES-A}}] - \nabla_{\boldsymbol{\theta}}L(\boldsymbol{\theta}) = \mathbf{0}$.*

Algorithm. Based on Eq. 6, we see that we can obtain unbiased gradient estimates from partial unrolls by: 1) *not resetting the particles* between unrolls, and 2) *accumulating the perturbations* $\boldsymbol{\xi}_t$ each particle has experienced over multiple unrolls.

Algorithm 1 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
σ, standard deviation of perturbations
α, learning rate for ES optimization
Initialize $\boldsymbol{s} = \boldsymbol{s}_0$

while true **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 2 Persistent evolution strategies (PES). Differences from ES are highlighted in purple.

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

Figure 2: **A comparison of vanilla ES and PES 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.

The resulting algorithm is simple to implement, requiring only minor modifications from vanilla ES. Algorithm 1 describes truncated ES applied to partial unrolls, where it suffers from short horizon bias. Algorithm 2 shows PES applied to the same problem, where it provides unbiased gradient estimates. Both algorithms (Fig. 2) are shown with antithetic sampling (perturbations are paired with their negations), which drastically reduces variance.

3 Experiments

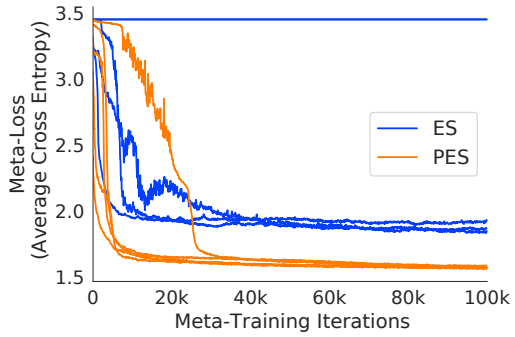


Figure 3: **Training learned optimizers**. We meta-train an MLP-based learned optimizer as described in [Metz *et al.*, 2019]. The learned optimizer is tasked to train a two hidden-layer, 128 unit, MLP on CIFAR-10 with a batch size of 128. Due to PES's unbiased nature, PES achieves both lower losses, and is more consistent across random initializations. Curves of the same color denote different initializations of a learned optimizer.

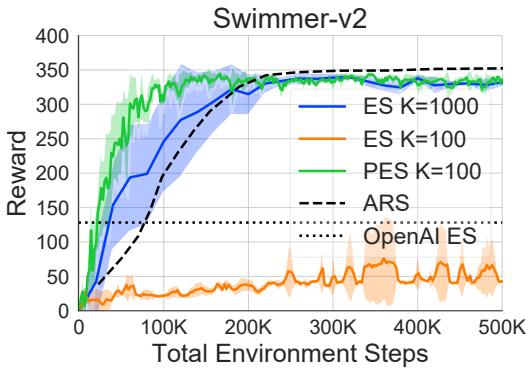


Figure 4: **Learning a policy for continuous control**. We train a linear policy on the Swimmer-v2 MuJoCo environment, following [Mania *et al.*, 2018]. We find that PES applied to truncated unrolls performs similarly to ES applied to full episodes, while truncated ES fails due to bias. We plot the ARS V1 result from [Mania *et al.*, 2018] (dashed curve) to show that our full-unroll baseline is comparable to theirs. The dotted line shows the maximum reward reported for the ES approach in [Salimans *et al.*, 2017], which does not solve the Swimmer task.

In experiments, we use PES to meta-train a learned optimizer (Figure 3), and learn a policy for continuous control (Figure 4). We additionally demonstrate the use of PES for hyperparameter optimization, where we show that PES performs well when the meta-loss has many local minima (Figure 5), does not suffer from truncation bias (Figure 6a), can be applied to non-differentiable objectives (Figure 6b), and can be used to optimize many hyperparameters (both continuous

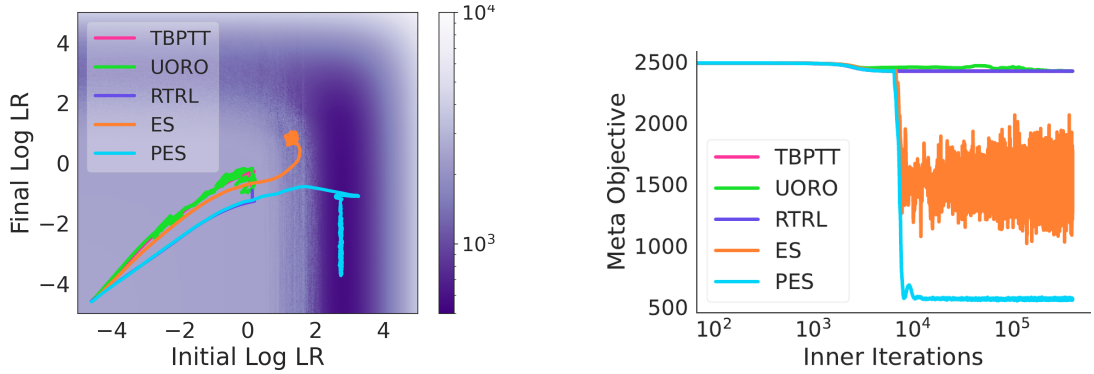


Figure 5: **The meta-objective surface (left), and meta-objective vs inner problem timesteps (right)**, for a 2D regression problem with many local minima to which truncated methods could converge. We plot meta-optimization trajectories for TBPTT, UORO, RTRL, ES, and PES starting from the same initialization, $(-4.5, -4.5)$ in log-space. All techniques except PES either suffer from truncation bias, or become stuck due to high-frequency structure in the meta-objective surface. PES is both unbiased, and smooths the outer-objective removing high-frequency structure.

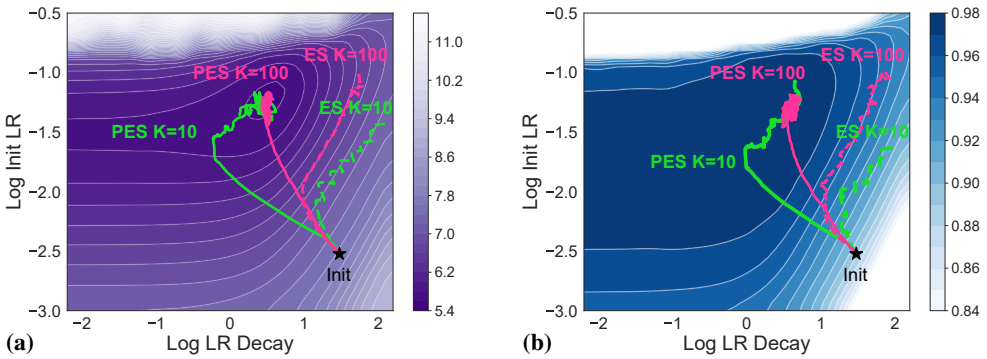


Figure 6: **Meta-optimization of a learning rate schedule for an MLP on MNIST.** We tune the initial learning rate and decay factor, both parameterized in log-space. Here we show the meta-loss landscape, and the optimization trajectories taken by ES and PES, for unroll lengths $K \in \{10, 100\}$. The meta-objective in **(a)** is the training loss, while the meta-objective in **(b)** is the validation accuracy. In both visualizations, darker colors denote better values. We see that ES converges to a suboptimal region of the hyperparameter space due to truncation bias, while PES finds the correct solution. Because PES only requires function evaluations and not gradients, it can optimize non-differentiable objectives such as accuracy rather than loss. This is demonstrated in subfigure **(b)**.

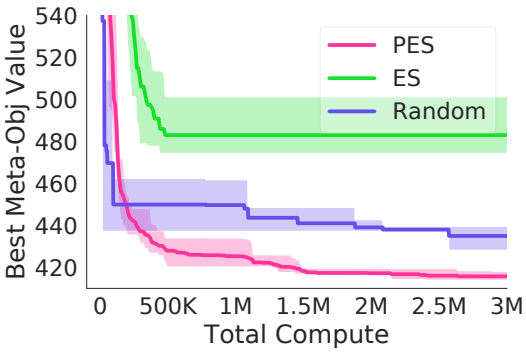


Figure 7: **Meta-optimization of per-parameter-block learning rates and momentum coefficients (29 hyperparameters total).** We tuned both continuous and discrete hyperparameters: the number of units per hidden layer (discrete architectural hyperparameters) and per-parameter-block learning rates and momentum coefficients (continuous hyperparameters). We trained a 5-hidden-layer MLP (6 layers including the output layer mapping to logits) on FashionMNIST, yielding 29 hyperparameters in total.

and discrete) simultaneously (Figure 7). A detailed description of each experimental setup is given in the full paper, [Vicol *et al.*, 2021]. A tutorial Colab notebook demonstrating the technique, and reproducing some of the experimental results, is available at this link.

4 Conclusion

We introduce a method called Persistent Evolution Strategies (PES), which divides a computation graph into a series of truncated unrolls, and performs an evolution strategies-based update after each unroll. PES eliminates bias from these truncations by accumulating correction terms over the entire sequence of unrolls. PES allows for rapid parameter updates, has low memory usage, is unbiased, and has reasonable variance characteristics. We show that PES is broadly applicable, with experiments demonstrating its use in hyperparameter optimization, reinforcement learning, and meta-training of learned optimizers.

Acknowledgements

This work was done while on internship at Google. We thank Sergey Ioffe and Niru Maheswaranathan for very helpful discussions and feedback on the paper.

References

[Beatson and Adams, 2019] Alex Beatson and Ryan P Adams. Efficient optimization of loops and limits with randomized telescoping sums. *arXiv preprint arXiv:1905.07006*, 2019.

[Benzing *et al.*, 2019] Frederik Benzing, Marcelo Matheus Gauy, Asier Mujika, Anders Martinsson, and Angelika Steger. Optimal Kronecker-sum approximation of real time recurrent learning. *arXiv preprint arXiv:1902.03993*, 2019.

[Cui *et al.*, 2018] Xiaodong Cui, Wei Zhang, Zoltán Tüske, and Michael Picheny. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 6048–6058, 2018.

[Franceschi *et al.*, 2018] Luca Franceschi, Paolo Frasconi, Saverio Salzo, Riccardo Grazzi, and Massimilano Pontil. Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv:1806.04910*, 2018.

[Ha and Schmidhuber, 2018] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[Ha, 2020] David Ha. Neuroevolution for deep reinforcement learning problems. In *Genetic and Evolutionary Computation Conference Companion*, pages 404–427, 2020.

[Kingma and Ba, 2015] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[Mania *et al.*, 2018] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

[Metz *et al.*, 2018] Luke Metz, Niru Maheswaranathan, Brian Cheung, and Jascha Sohl-Dickstein. Meta-learning update rules for unsupervised representation learning. *arXiv preprint arXiv:1804.00222*, 2018.

[Metz *et al.*, 2019] Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *International Conference on Machine Learning*, pages 4556–4565, 2019.

[Mujika *et al.*, 2018] Asier Mujika, Florian Meier, and Angelika Steger. Approximating real-time recurrent learning with random Kronecker factors. In *Advances in Neural Information Processing Systems*, pages 6594–6603, 2018.

[Parmas *et al.*, 2018] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. PIPPS: Flexible model-based policy search robust to the curse of chaos. In *International Conference on Machine Learning*, pages 4062–4071, 2018.

[Pascanu *et al.*, 2013] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[Salimans *et al.*, 2017] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

[Staines and Barber, 2012] Joe Staines and David Barber. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

[Tallec and Ollivier, 2017] Corentin Tallec and Yann Ollivier. Unbiased online recurrent optimization. *arXiv preprint arXiv:1702.05043*, 2017.

[Tieleman and Hinton, 2012] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5—RMSprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[Vicol *et al.*, 2021] Paul Vicol, Luke Metz, and Jascha Sohl-Dickstein. Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies. In *International Conference on Machine Learning*, pages 10553–10563, 2021.

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

[Wichrowska *et al.*, 2017] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. *arXiv preprint arXiv:1703.04813*, 2017.

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

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