Unbiased Gradient Estimation in Unrolled Computation Graphs with Persistent Evolution Strategies

Paul Vicol [1 2] **Luke Metz** [2] **Jascha Sohl-Dickstein** [2]

Abstract

Unrolled computation graphs arise in many scenarios, including training RNNs, tuning hyperparameters through unrolled optimization, and training learned optimizers. Current approaches to optimizing parameters in such 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 characteristics. We experimentally demonstrate the advantages of PES compared to several other methods for gradient estimation on synthetic tasks, and show its applicability to training learned optimizers and tuning hyperparameters.

1. Introduction

Unrolled computation graphs arise in many scenarios in machine learning, including when training RNNs (Williams & Peng, 1990), tuning hyperparameters through unrolled computation graphs (Baydin et al., 2017; Domke, 2012; Maclaurin et al., 2015; Wu et al., 2018; Franceschi et al., 2017; Donini et al., 2019; Franceschi et al., 2018; Liu et al., 2018; Shaban et al., 2019), and training learned optimizers (Li & Malik, 2016; 2017; Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2018; 2019; 2020; met). Many methods exist for computing gradients in such computation graphs, including those based on reverse-mode (Williams & Peng, 1990; Tallec & Ollivier, 2017b; Aicher et al., 2019; Grefenstette et al., 2019) and forward-mode (Williams & Zipser, 1989; Tallec & Ollivier, 2017a; Mujika et al., 2018; Benzing et al., 2019; Marschall et al., 2019; Menick et al., 2020) gradient accumulation. These methods have different tradeoffs with respect to compute, memory, and gradient variance.

Backpropagation through time involves backpropagating through a full unrolled sequence (e.g. of length T) for each parameter update. Unrolling a model over full sequences faces several difficulties: 1) the memory cost scales linearly with the unroll length, because we need to store intermediate activations for backprop (though this can be reduced at the cost of additional compute (Dauvergne & Hascoët, 2006; Chen et al., 2016)); 2) we only perform a single parameter update after each full unroll, which is computationally expensive and introduces large latency between parameter updates; 3) long unrolls can lead to exploding or vanishing gradients (Pascanu et al., 2013), and chaotic and poorly conditioned loss landscapes (Pearlmutter, 1996; Maclaurin et al., 2015; Parmas et al., 2018; Metz et al., 2019). This is especially true in meta-learning (Metz et al., 2019).

The most commonly-used technique to alleviate these issues is truncated backprop through time (TBPTT) (Werbos, 1990; Tallec & Ollivier, 2017b), which splits the full sequence into shorter sub-sequences and performs a backprop update after processing each sub-sequence. However, a critical drawback of TBPTT is that it yields biased gradients, that can severely impact training (e.g. only taking into account short-term dependencies). To address the poorly conditioned loss surfaces that often result from sequential computation, it can additionally 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. ES works well on pathological meta-optimization loss surfaces (Metz et al., 2019); however, due to the computational expense of running full unrolls, ES can only practically be applied in a truncated fashion, introducing bias.

An alternative to BPTT is real-time recurrent learning (RTRL), which performs forward gradient accumulation (Williams & Zipser, 1989). RTRL enables online parameter updates (after each partial unroll) and does not suf-

[1]University of Toronto [2]Google Brain. Correspondence to: Paul Vicol <pvicol@cs.toronto.edu>.

In submission to the 38^{th} International Conference on Machine Learning, PMLR 139, 2021. Copyright 2021 by the author(s).

fer from truncation bias; however, its memory and compute requirements render it intractable for large-scale problems. Many approximations to RTRL have been proposed (Tallec & Ollivier, 2017a; 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.

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 vanilla ES—which yields an unbiased estimate of the gradient even when using truncated sequences. PES is simple to implement, and because it is an evolution strategies-based approach, it retains desirable characteristics such as being trivially parallelizable, memory efficient, and broadly applicable to many different types of problems, including to non-differentiable target functions.

Contributions

- We introduce a method called Persistent Evolution Strategies (PES) to obtain unbiased gradient estimates for the parameters of an unrolled system from *partial unrolls* of the system.
- We prove that PES is an unbiased gradient estimate for a smoothed version of the loss, and an unbiased estimate of the true gradient for quadratic losses. We provide theoretical and empirical analyses of its variance.
- We demonstrate the applicability of PES in several illustrative scenarios: 1) we apply PES to tune hyperparameters including learning rates and momentums, by estimating hypergradients through partial unrolls of optimization algorithms; 2) we use PES to meta-train a learned optimizer; 3) we use PES to learn policy parameters for a continuous control task.

2. Background

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. At each step t we define a loss $L_t(\boldsymbol{s}_t; \boldsymbol{\theta})$, and we consider the objective function (for optimizing the parameters $\boldsymbol{\theta}$) to be the sum of per-timestep losses:

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

This setup is quite general, even encompassing situations where we want to consider only a final loss at step T, which can be expressed using a telescoping sum of loss differences (Beatson & Adams, 2019). [1] Instances of this problem setup include training RNNs, training learned optimizers, learning policies for control tasks and unrolled optimization, as illustrated in Figure 1. An overview of notation is presented in Appendix A.

Unrolled Optimization. Optimization algorithms can be unrolled to yield computation graphs, in which the nodes are the model parameters at successive optimization steps. Estimating gradients through unrolled optimization has been used to tune hyperparameters (Domke, 2012; Maclaurin et al., 2015; Baydin et al., 2017; Donini et al., 2019; Franceschi et al., 2017) and train learned optimizers (Li & Malik, 2016; 2017; Andrychowicz et al., 2016; Wichrowska et al., 2017; Metz et al., 2019; 2020; met; Metz et al., 2018).

Truncation Bias. Truncation, or short horizon, bias poses a major challenge when unrolled optimization is decomposed into a sequence of short sequential unrolls of length $K \ll T$. These challenges have been demonstrated in gradient-based hyperparameter optimization (Wu et al., 2018) and in the training of learned optimizers (Metz et al., 2019). Approaches to mitigating short horizon bias are an area of active research (Micaelli & Storkey, 2020).

Smoothing. Unrolling optimization for many steps can lead to pathological meta-loss 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. One 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 & Barber, 2012; Metz et al., 2019). Conveniently, ES provides an unbiased estimate of the gradient of this smoothed objective. However, the ES estimate remains biased when computed on truncations.

Evolution Strategies. Evolution Strategies (ES) (Rechenberg, 1973; Nesterov & Spokoiny, 2017) refers to a family of methods for estimating a descent direction for arbitrary black-box functions using stochastic finite differences. Since ES only requires function evaluations and not gradients, it is a zeroth-order optimization method. The vanilla ES estimator is defined as:

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

[1] For more details on telescoping sums, see Appendix D.

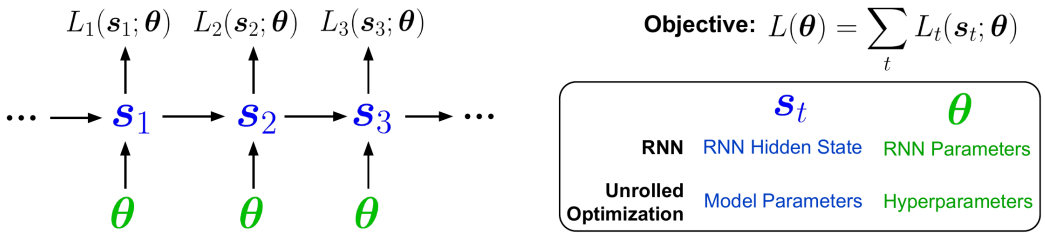

Figure 1. **An unrolled computation graph**, illustrating how both RNNs and unrolled optimization can be described using Equations 1 and 2. In RNN training, $\boldsymbol{s}_t$ is the hidden state of the RNN, $L_t(\cdot)$ is the prediction cross entropy at each timestep, $\boldsymbol{\theta}$ refers to the RNN parameters, and f corresponds to the forward pass of the RNN, that takes an input and the previous hidden state ($\boldsymbol{s}_t$) and returns a new hidden state ($\boldsymbol{s}_{t+1}$). 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 & Hinton, 2012), or Adam (Kingma & Ba, 2015).

where $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(0, \sigma^2 I)$. ES is trivially parallelizable, and thus highly scalable—it has seen renewed interest in recent years as a viable optimization algorithm for reinforcement learning among other black-box problems (Salimans et al., 2017; Mania et al., 2018; Ha & Schmidhuber, 2018; Houthooft et al., 2018; Cui et al., 2018; Ha, 2020). The estimator in Eq. 3 has high variance, and thus many variance reduction techniques have been proposed, including control variates (Tang et al., 2020) and antithetic sampling (Owen, 2013). Antithetic sampling involves using pairs of function evaluations $\boldsymbol{\theta} + \boldsymbol{\epsilon}$ and $\boldsymbol{\theta} - \boldsymbol{\epsilon}$, yielding the following estimator:

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

where N is even, and $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(0, \sigma^2 I)$. Several methods have been proposed to improve the search space for ES, including covariance matrix adaptation ES (CMA-ES) (Hansen, 2016) and Guided ES (Maheswaranathan et al., 2018). A limitation of ES is that applying it to full unrolls is often computationally costly (as we only make one update to the system parameters every full unroll), while applying ES to partial unrolls suffers from truncation bias similarly to TBPTT. In contrast, PES allows computation of gradients from partial updates without incurring truncation bias.

Hysteresis. Any approach that performs online parameter updates, including Real Time Recurrent Learning (RTRL) and its approximations, will suffer from hysteresis, which refers to the dependence of the state of a system on its history. This is due to the fact that if we update $\boldsymbol{\theta}$, then any accumulated state (e.g. in the case of RTRL, the accumulated Jacobian $\frac{d\boldsymbol{s}_t}{d\boldsymbol{\theta}}$) will be incorrect because it is computed from previous values of $\boldsymbol{\theta}$. To eliminate hysteresis completely, one would need to run the full sequence for a given problem for each parameter update, which is often prohibitively expensive. In practice, hysteresis can be mitigated by using sufficiently small learning rates; this introduces a tradeoff between training stability and training speed.

3. Related Work

In this section, we discuss additional related work on online learning algorithms, and on one special class of unrolled optimization problems: hyperparameter optimization (HO). Table 1 compares several approaches to gradient estimation in unrolled computation graphs, with respect to compute, memory, parallelization, unbiasedness, and smoothing. In addition, Table 3 in Appendix B provides a comparison of many of the HO algorithms mentioned in this section.

RTRL. Real-time recurrent learning (RTRL) performs forward-mode gradient accumulation: it does not require storage of past states, but requires matrix-matrix products and storage of a matrix G_t of size $\dim(\boldsymbol{s}_t) \times \dim(\boldsymbol{\theta})$. When $\dim(\boldsymbol{\theta})$ is large, as in RNN training, the cost of storing G_t and the cost of computing the required matrix-matrix products is prohibitive. Several approaches propose efficient variants of RTRL based on cheaper, noisy approximations of G_t. Unbiased Online Recurrent Optimization (UORO) (Tallec & Ollivier, 2017a) uses an unbiased rank-1 approximation to the full matrix; Kronecker-Factored RTRL (KF-RTRL) (Mujika et al., 2018) uses a Kronecker product decomposition to approximate the RTRL update for a class of RNNs; and Optimal Kronecker Sum Approximation (OK) (Benzing et al., 2019) uses a similar approximation but with the lowest possible variance among methods within an approximation family. Cooijmans & Martens (2019) also draw a connection between UORO and REINFORCE applied to estimate the gradient of an RNN by injecting noise into the hidden states. In contrast, PES injects noise into the parameters.

Hyperparameter Optimization (HO) There are three main approaches that can be categorized based on the types of problem-specific information used: 1) *black-box* approaches that do not consider the internal structure of the objective L; 2) *gray-box* approaches that make use of the

Table 1. **Comparison of approaches for learning parameters in unrolled computation graphs**. S is the size of the system state (e.g. the RNN hidden state dimension, or in the case of hyperparameter optimization the inner-problem's weight dimensionality and potentially the optimizer state; P is the dimensionality of $\boldsymbol{\theta}$; T is the total number of steps in a sequence/unroll; K is the truncation length; and N is the number of samples (also called *particles*) used for the reparameterization gradient and in ES-based algorithms; F and B are the costs of a forward and backward pass, respectively; terms in purple denote computation/memory that can be split across parallel workers. See Appendix J for details.

Method	Compute	Memory	Parallel	Unbiased	Optimize Non-Diff.	Smoothed
BPTT (Rumelhart et al., 1985)	$T(F+B)$	TS	✗	✓	✗	✗
TBPTT (Williams & Peng, 1990)	$K(F+B)$	KS	✗	✗	✗	✗
ARTBP (Tallec & Ollivier, 2017b)	$K(F+B)$	KS	✗	✓	✗	✗
RTRL (Williams & Zipser, 1989)	$PS^2 + S(F+B)$	$SP + S^2$	✗	✓	✗	✗
UORO (Tallec & Ollivier, 2017a)	$F + B + S^2 + P$	$S + P$	✗	✓	✗	✗
Reparam. (Metz et al., 2019)	$NT(F+B)$	NTS	✓	✓	✗	✓
ES (Rechenberg, 1973)	NTF	NS	✓	✓	✓	✓
Trunc. ES (Metz et al., 2019)	NKF	NS	✓	✗	✓	✓
PES (Ours)	NKF	$N(S+P)$	✓	✓	✓	✓
PES + Analytic (Ours)	$NKF + K(F+B)$	$N(S+P) + (K+1)S$	✓	✓	✗	✓

fact that the objective is the result of an iterative optimization procedure (e.g. by using the validation performance of a model); and 3) *gradient-based* approaches that require access to the exact functional form of the objective L, and that require the objective to be differentiable in the hyperparameters. Black-box approaches include grid search, random search (Bergstra & Bengio, 2012), Bayesian optimization (BO) (Snoek et al., 2012), and ES (Salimans et al., 2017; Metz et al., 2019). Gray-box approaches include Freeze-Thaw BO (Swersky et al., 2014), successive halving (Jamieson & Talwalkar, 2016), Hyperband (Li et al., 2017), Population-Based Training (Jaderberg et al., 2017), and hypernetwork-based approaches to HO (Lorraine & Duvenaud, 2018; MacKay et al., 2019).

A key advantage of gradient-based approaches is that they scale to high-dimensional hyperparameters (e.g. millions of hyperparameters) (Lorraine et al., 2020). Maclaurin et al. (2015) differentiate through unrolled optimization to tune many hyperparameters including learning rates and weight decay coefficients. These methods can perform poorly, however, when the underlying meta-loss is not smooth. Additionally they cannot optimize non-differentiable objectives, for example accuracy rather than loss.

PES can be considered a gray-box approach as it does not require the objective to be differentiable like gradient-based approaches, but it does take into account the iterative optimization of the inner problem.

4. Persistent Evolution Strategies

In this section, we introduce a method to obtain unbiased gradient estimates from partial unrolls of a computation graph, called Persistent Evolution Strategies (PES). First, we derive the PES gradient estimator, prove that it is unbiased, and present a practical algorithm (Algorithm 2). Then we discuss the variance characteristics of PES, both theoretically and empirically.

Derivation.[2] 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 $\Theta = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_T)^\top$, 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 $L_t(\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_t)$ 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 $\frac{\partial L(\Theta)}{\partial \operatorname{vec}(\Theta)}$,

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

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

$$= \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}} \left[\left(\sum_{\tau=1}^{T} \boldsymbol{\epsilon}_\tau\right) L(\Theta + \boldsymbol{\epsilon})\right],$$

where $\otimes$ denotes the Kronecker product, $\boldsymbol{\epsilon} = (\boldsymbol{\epsilon}_1, \dots, \boldsymbol{\epsilon}_T)^\top$ is a matrix of perturbations $\boldsymbol{\epsilon}_t$ to be added to the $\boldsymbol{\theta}_t$ at each

[2] See Appendix E for an expanded derivation, and Appendix I for an alternate derivation using stochastic computation graphs (Schulman et al., 2015).

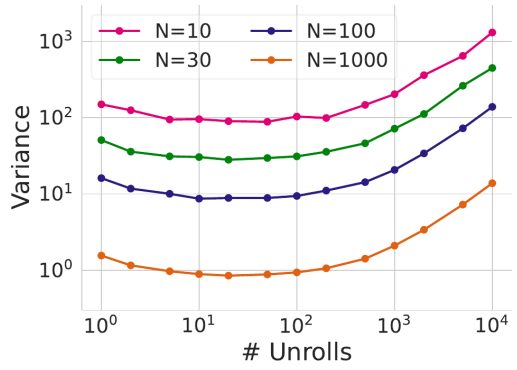

Figure 2. **Empirical variance measurements.** Variance of the PES gradient estimate for different numbers of particles (N) as a function of number of unrolls for a language modeling task.

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,

$$\begin{aligned}\boldsymbol{g}^{\text{PES}} &= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\left(\sum_{\tau=1}^{T}\boldsymbol{\epsilon}_\tau\right)\sum_{t=1}^{T}L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right] \\ &= \frac{1}{\sigma^2}\mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\left(\sum_{\tau=1}^{t}\boldsymbol{\epsilon}_\tau\right)L_t\left(\Theta+\boldsymbol{\epsilon}\right)\right] \qquad (4)\\ &= \mathbb{E}_{\boldsymbol{\epsilon}}\left[\sum_{t=1}^{T}\hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}}\right], \qquad (5)\end{aligned}$$

$$\hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}} = \frac{1}{\sigma^2}\boldsymbol{\xi}_t L_t\left(\boldsymbol{\theta}_1+\boldsymbol{\epsilon}_1,\ldots,\boldsymbol{\theta}_t+\boldsymbol{\epsilon}_t\right). \qquad (6)$$

where $\boldsymbol{\xi}_t = \sum_{\tau=1}^{t}\boldsymbol{\epsilon}_\tau$, Equation 4 relies on $L_t(\cdot)$ being independent of $\boldsymbol{\epsilon}_\tau$ for $\tau > t$, and Equation 6 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 5,

$$\hat{\boldsymbol{g}}^{\text{PES}} = \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\hat{\boldsymbol{g}}^{\text{PES}}_{t,\boldsymbol{\epsilon}^{(i)}} \qquad (7)$$

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.

PES is Unbiased for Quadratic Losses. See Appendix F for a proof of the following Statement 4.1.

Statement 4.1 (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:* $L(\Theta+\boldsymbol{\epsilon}) = L(\Theta) + \text{vec}(\boldsymbol{\epsilon})^\top\nabla_{\text{vec}(\Theta)}L(\Theta) + \frac{1}{2}\text{vec}(\boldsymbol{\epsilon})^\top\nabla^2_{\text{vec}(\Theta)}L(\Theta)\,\text{vec}(\boldsymbol{\epsilon})$. *Consider the PES estimator (using antithetic sampling) below:*

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

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

Algorithm. Based on Eq. 7, 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. 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. 3) are shown with antithetic sampling (perturbations are paired with their negations), which drastically reduces variance.

Variance Analysis.[3] We use the total variance, $\text{tr}(\text{Var}(\hat{\boldsymbol{g}}^{\text{PES}}))$, to quantify the variance of the estimator. We provide a full derivation of the variance in Appendix G, and here we present some takeaways. The variance of the antithetic form of the PES estimator from Eq. 7 is:

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}}^{\text{PES}})) = \sum_{t=1}^{T}||\boldsymbol{g}_t||^2\,(Pt+1) \qquad (8)$$

where $\boldsymbol{g}_t = \nabla L_t(\boldsymbol{\theta})$ is the true gradient at step t, and P is the dimensionality of $\boldsymbol{\theta}$. To understand the properties of the variance, consider two potential scenarios. For the first scenario, if we assume that the gradients for each unroll are i.i.d., then $\mathbb{E}[||\boldsymbol{g}_t||^2] = \frac{1}{T}\mathbb{E}[||\boldsymbol{g}||^2]$, and we have:

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}}^{\text{PES}})) = ||\boldsymbol{g}||^2\left(\frac{1}{2}PT+\frac{1}{2}P+1\right) \qquad (9)$$

For the second scenario, if we assume that the gradients for each unroll are identical to each other, then $||\boldsymbol{g}_t||^2 = \frac{1}{T^2}||\boldsymbol{g}||^2$, and we have:

$$\text{tr}(\text{Var}(\hat{\boldsymbol{g}}^{\text{PES}})) = ||\boldsymbol{g}||^2\left(\frac{1}{2}P+\frac{P}{2T}+\frac{1}{T}\right) \qquad (10)$$

It is possible (though atypical) for the $\boldsymbol{g}_t$ to have variance larger than predicted by either of these two reasonable scenarios. For instance, if $||\boldsymbol{g}_t||$ remains roughly constant even as the number of unrolls is increased, then variance grows

[3] While submitting the camera ready of this paper, we discovered a mistake in our variance derivation. We do not believe it will have a qualitative effect (especially given the experimental support for the current scaling behavior), but please see the arXiv version of the paper for an updated discussion of variance.

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 $\boldsymbol{s} = \boldsymbol{s}_0$

repeat
  $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \mathbf{0}$
  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
  $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{ES}}$
  $\boldsymbol{s} \leftarrow \text{unroll}(\boldsymbol{s}, \boldsymbol{\theta}, K)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{ES}}$

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 $i \in \{1, \ldots, N\}$
repeat
  $\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \mathbf{0}$
  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
  $\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{PES}}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{PES}}$

Figure 3. **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. See Appendix K for diagrammatic representations of these algorithms.

like $\mathcal{O}(T^2)$.

To investigate the variance characteristics of PES empirically, we computed the variance in a toy setting. We used an LSTM with 5 hidden units and 5-dimensional embeddings, for character-level language modeling on the Penn Treebank corpus (Marcus et al., 1993) (with 50 tokens). We measured the variance of the gradient of a fixed sequence of 10^4 characters. The ground-truth gradient was computed using vanilla ES with 500k particles over the full sequence (without truncation). Figure 2 shows the variance of the PES gradient estimate using different numbers of unrolls, ranging from 1 (a single unroll for the full sequence) to 10^4 (one unroll per input token). We plot the variance normalized by the squared norm of the ground-truth gradient. We observe an initial *drop* in variance predicted by Equation 10, and then a linear growth as predicted by Equation 9. Empirical variance measurements for other plausible scenarios are presented in Figure 16 (Appendix G).

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. In Appendix H, we show how we can incorporate the analytic gradient in the ES estimate for $\frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}}$, deriving the following estimator:

$$\frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}} \approx \frac{1}{\sigma^2} \mathbb{E}_{\boldsymbol{\epsilon}} \left[\left(\sum_{\tau < t} \boldsymbol{\epsilon}_\tau \right) \left(L_t(\Theta + \boldsymbol{\epsilon}) - \boldsymbol{\epsilon}_t^\top \boldsymbol{p}_t \right) \right] + \boldsymbol{p}_t \tag{11}$$

where $\boldsymbol{p}_t = \frac{\partial L_t(\Theta)}{\partial \boldsymbol{\theta}_t}$. We call the resulting estimator PES+Analytic. In Appendix H we describe the implementation of this estimator (Algorithm 4), which requires a few simple changes from the standard PES estimator. We also provide empirical variance measurements for PES+Analytic, using the same setup as was used for Figure 2; we found that it can reduce variance by 1-2 orders of magnitude, given the same number of particles as PES.

5. Experiments

First, we demonstrate via a toy experiment that PES does not suffer from truncation bias, allowing it to converge to correct solutions that are not found by TBPTT or truncated ES. Then, we apply PES to several illustrative scenarios: we use PES to meta-train a learned optimizer, learn a policy for continuous control, and optimize hyperparameters. All experiments used JAX (Bradbury et al., 2018). A simplified code snippet implementing PES is provided in Appendix M.

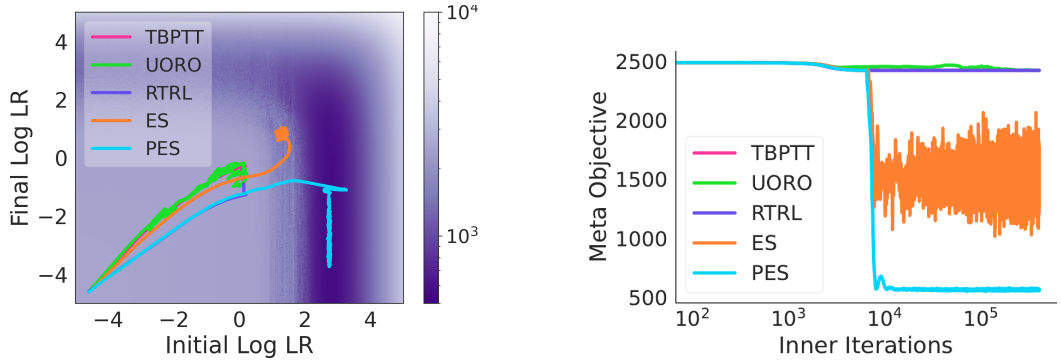

Figure 4. **The meta-objective surface (left), and meta-objective vs inner problem timesteps (right)**, for the 2D regression problem in Section 5.4. We plot meta-optimization trajectories for TBPTT, UORO, RTRL, ES, and PES starting from the same initialization, $(-4.5, -4.5)$. All techniques except 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. Ablations over the truncation length and number of particles for this task are provided in Appendix L.

5.1. Influence Balancing

To demonstrate the lack of truncation bias of PES in a toy setting, we use the influence balancing task introduced by Tallec & Ollivier (2017a). This simple task is particularly sensitive to short-horizon bias, as the gradient for the single parameter $\theta \in \mathbb{R}$ has the wrong sign when estimated from short unrolls. See Appendix C.2 for more details.

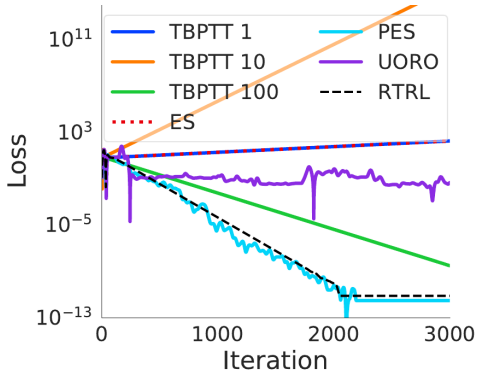

Figure 5. **Loss curves for the influence balancing task**. TBPTT with short truncations diverges, while PES performs nearly identically to exact RTRL. See Section 5.1 for experiment details.

We use vanilla SGD to update θ, with gradient estimates derived from TBPTT with different truncation horizons, exact RTRL, UORO, and PES. TBPTT does not converge with short truncations $K \in \{1, 10\}$; it requires much longer truncations $K = 100$ to move in the right direction. In Figure 5, we show that PES achieves nearly identical performance to exact RTRL. UORO reaches the same performance as RTRL after approximately $30k$ iterations. Note that the purpose of this experiment is to demonstrate that PES is unbiased, and is able to match the performance of exact RTRL given sufficiently many particles ($N = 10^3$) to reduce variance; it is not intended as a comparison of total compute.

5.2. Learned Optimizer Meta-Optimization

In this section we demonstrate PES's applicability for learned optimizer training. We meta-train an MLP-based learned optimizer as described in Metz et al. (2019). This optimizer is used to train a two hidden-layer, 128 unit, MLP on CIFAR-10 with a batch size of 128. Our meta-objective is the average training loss. We train with a total number of inner-steps of $T = 1000$ and a truncation length of $K = 4$, using both PES and truncated ES.

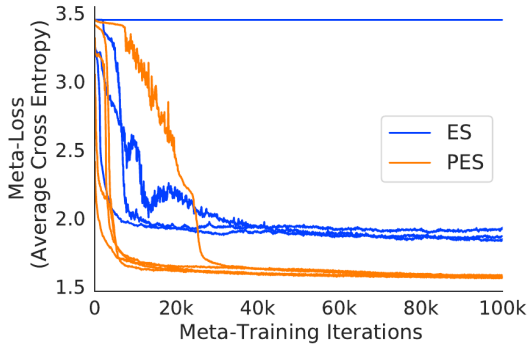

Figure 6. **Training learned optimizers**. We find that PES achieves better performance compared to truncated ES. Curves of the same color denote different initializations of the learned optimizer. See Section 5.2 for details.

We outer-train with Adam, using a learning rate of 10^{-4} selected via grid search over half-orders of magnitude for each method independently. We use gradient clipping of 3 applied to each gradient coordinate. We outer-train on 8 TPUv2 cores with asynchronous, batched updates of size 16. To evaluate, we compute the meta-loss averaged over 20 inner initializations over the course of meta-training. Results can be found in Figure 6. Due to PES's unbiased nature, PES achieves both lower losses, and is more consistent across random initializations of the learned optimizer.

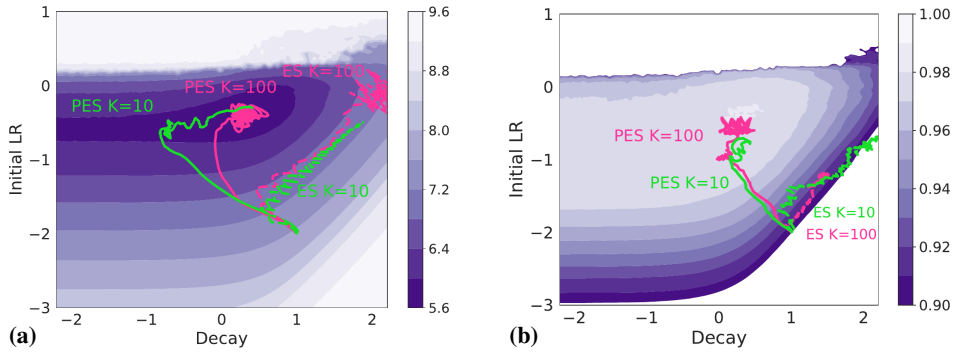

Figure 7. **Meta-optimization of a learning rate schedule for an MLP on MNIST.** Here we show the meta-loss landscape, and the optimization trajectories taken by both ES and PES, for unroll lengths K of 10 and 100, for a meta-objective (shown in color) of *(a)* training loss (darker color is better) and *(b)* validation accuracy (lighter color is better). Note that most gradient-based approaches are unable to target accuracy.

5.3. Learning a Continuous Control Policy

Recent work (Salimans et al., 2017; Mania et al., 2018) has shown that ES-based algorithms can be a viable alternative to more complex RL algorithms. ES optimizes the parameters of a policy directly, by sampling parameters from a distribution, running an episode, and estimating the gradient; this is in contrast to standard RL algorithms that sample actions from a distribution output by a policy. Here, we

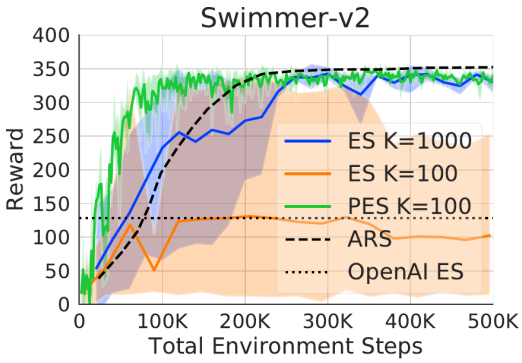

Figure 8. **Learning a policy for continuous control**. We find that PES is more efficient than ES applied to full episodes, while truncated ES fails due to bias. We plot the ARS V1 result from Mania et al. (2018) (dashed curve) to show that our full-unroll baseline is comparable to theirs. The dotted line shows the maximum reward reported for the ES approach in Salimans et al. (2017), which does not solve the Swimmer task. See Section 5.3 for details.

demonstrate that PES can be used to train a policy for a continuous control problem using partial unrolls, improving on the efficiency of vanilla ES typically applied to full unrolls. We train a linear policy on the Swimmer-v2 MuJoCo environment, following Mania et al. (2018). For PES, the objective for each partial unroll is the sum of rewards over that unroll. We also applied vanilla ES to the partial unrolls to demonstrate that this naïve strategy does not work—truncation bias occurs for these control problems as well. Figure 8 compares vanilla ES applied to full episodes, ES applied to partial episodes, PES applied to partial episodes, and variants of ES from Mania et al. (2018) and Salimans et al. (2017). We see that PES reaches the same performance as full-unroll ES in fewer total environment steps.

5.4. Hyperparameter Optimization

In this section we provide evidence that PES can be used for hyperparameter optimization across four different problems. We show that PES performs well when the meta-loss has many local minima, does not suffer from truncation bias, can be applied to non-differentiable objectives, and can be used to optimize many hyperparameters simultaneously.

Toy 2D Regression. First, we used PES to meta-optimize a learning rate schedule for a toy 2D regression problem that has one global minimum, but many local minima to which truncated gradient methods could converge. The inner optimization trajectories for different values of the outer-parameters are shown in Appendix C. We used a linear learning rate schedule parameterized by the initial and final log-learning rates, θ_0 and θ_1, respectively: $\alpha_t = \left(1 - \frac{t}{T}\right) e^{\theta_0} + \frac{t}{T} e^{\theta_1}$. In Figure 4 we compare TBPTT, UORO, RTRL, ES, and PES applied to this meta-optimization task.

MNIST MLP. Next, we used PES to meta-learn a learning rate schedule for an MLP classifier on MNIST. 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. The full unrolled inner problem consists of $T = 5000$ optimization steps, and we consider using 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 losses over the inner optimization trajectory. In Figure 7(a) we see that ES converges to a suboptimal region of the hyperparameter space due to truncation bias, while

PES finds the correct solution.

Targeting Validation Accuracy. Because PES only requires function evaluations and not gradients, it can optimize non-differentiable objectives such as accuracy rather than loss. We demonstrate this by tuning the same parameterization of learning rate schedule as in the previous section, but using the accuracy on the MNIST validation set as the meta-objective. Figure 7(b) compares the meta-optimization trajectories of ES and PES on the validation accuracy meta-objective; again ES is biased and fails to converge to the right solution, while PES works well.

Tuning Many Hyperparameters. Here, we show that PES can tune several hyperparameters simultaneously, and achieves better performance than random search with an uninformative search space, using less compute. We tuned separate learning rates and momentum coefficients for each weight matrix and bias term of a four-layer MLP with 100 hidden units per layer, yielding 20 hyperparameters. Figure 9 compares the best objective values achieved with a given compute budget by random search, ES, and PES, each evaluated using four random seeds.

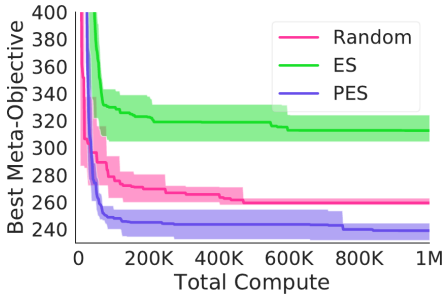

Figure 9. **Meta-optimization of per-parameter-block learning rates and momentum coefficients (20 hyperparameters total).**

We measured the compute as the number of inner optimization iterations performed across all parallel runs in random search, and across all particles in ES and PES. PES substantially outperforms ES and random search, achieving lower loss with a fraction of the compute.

6. Conclusion

We introduced a method for unbiased gradient estimation in unrolled computation graphs, called Persistent Evolution Strategies (PES). PES obtains gradients from truncated unrolls—which speeds up optimization by allowing for frequent parameter updates—while not suffering from truncation bias that affects many competing approaches. We show that PES is broadly applicable, with experiments demonstrating its application to an RNN-like task, hyperparameter optimization, reinforcement learning, and meta-training of learned optimizers.

Acknowledgements

We thank Sergey Ioffe and Niru Maheswaranathan for very helpful discussions and feedback on the paper.

References

Aicher, C., Foti, N. J., and Fox, E. B. Adaptively truncating backpropagation through time to control gradient bias. *arXiv preprint arXiv:1905.07473*, 2019.

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

Asuncion, A. and Newman, D. UCI machine learning repository, 2007.

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

Beatson, A. and Adams, R. P. Efficient optimization of loops and limits with randomized telescoping sums. *arXiv preprint arXiv:1905.07006*, 2019.

Benzing, F., Gauy, M. M., Mujika, A., Martinsson, A., and Steger, A. Optimal Kronecker-sum approximation of real time recurrent learning. *arXiv preprint arXiv:1902.03993*, 2019.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., and Wanderman-Milne, S. JAX: Composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

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

Cooijmans, T. and Martens, J. On the variance of unbiased online recurrent optimization. *arXiv preprint arXiv:1902.02405*, 2019.

Cui, X., Zhang, W., Tüske, Z., and Picheny, M. Evolutionary stochastic gradient descent for optimization of deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 6048–6058, 2018.

Dauvergne, B. and Hascoët, L. The data-flow equations of checkpointing in reverse automatic differentiation. In *International Conference on Computational Science*, pp. 566–573, 2006.

Domke, J. Generic methods for optimization-based modeling. In *Artificial Intelligence and Statistics*, pp. 318–326, 2012.

Donini, M., Franceschi, L., Pontil, M., Majumder, O., and Frasconi, P. Scheduling the learning rate via hypergradients: New insights and a new algorithm. *arXiv preprint arXiv:1910.08525*, 2019.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. Forward and reverse gradient-based hyperparameter optimization. In *International Conference on Machine Learning*, pp. 1165–1173, 2017.

Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. Bilevel programming for hyperparameter optimization and meta-learning. *arXiv preprint arXiv:1806.04910*, 2018.

Grefenstette, E., Amos, B., Yarats, D., Htut, P. M., Molchanov, A., Meier, F., Kiela, D., Cho, K., and Chintala, S. Generalized inner loop meta-learning. *arXiv preprint arXiv:1910.01727*, 2019.

Ha, D. Neuroevolution for deep reinforcement learning problems. In *Genetic and Evolutionary Computation Conference Companion*, pp. 404–427, 2020.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Hansen, N. The CMA evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

Houthooft, R., Chen, Y., Isola, P., Stadie, B., Wolski, F., Ho, O. J., and Abbeel, P. Evolved policy gradients. In *Advances in Neural Information Processing Systems*, pp. 5400–5409, 2018.

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

Jamieson, K. and Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In *International Conference on Artificial Intelligence and Statistics*, pp. 240–248, 2016.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kumar, M., Dahl, G. E., Vasudevan, V., and Norouzi, M. Parallel architecture and hyperparameter search via successive halving and classification. *arXiv preprint arXiv:1805.10255*, 2018.

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

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

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

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

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

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

MacKay, M., Vicol, P., Lorraine, J., Duvenaud, D., and Grosse, R. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *arXiv preprint arXiv:1903.03088*, 2019.

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

Maheswaranathan, N., Metz, L., Tucker, G., Choi, D., and Sohl-Dickstein, J. Guided evolutionary strategies: Augmenting random search with surrogate gradients. *arXiv preprint arXiv:1806.10230*, 2018.

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

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

Marschall, O., Cho, K., and Savin, C. A unified framework of online learning algorithms for training recurrent neural networks. *arXiv preprint arXiv:1907.02649*, 2019.

Menick, J., Elsen, E., Evci, U., Osindero, S., Simonyan, K., and Graves, A. A practical sparse approximation for real time recurrent learning. *arXiv preprint arXiv:2006.07232*, 2020.

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

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

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

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

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

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

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

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

Pascanu, R., Mikolov, T., and Bengio, Y. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pp. 1310–1318, 2013.

Pearlmutter, B. *An investigation of the gradient descent process in neural networks*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 1996.

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

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

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

Schulman, J., Heess, N., Weber, T., and Abbeel, P. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pp. 3528–3536, 2015.

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

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

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

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

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

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

Tang, Y., Choromanski, K., and Kucukelbir, A. Variance reduction for evolution strategies via structured control variates. In *International Conference on Artificial Intelligence and Statistics*, pp. 646–656, 2020.

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

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

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

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

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

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