Understanding and correcting pathologies in the training of learned optimizers

Luke Metz [1] **Niru Maheswaranathan** [1] **Jeremy Nixon** [1] **C. Daniel Freeman** [1] **Jascha Sohl-Dickstein** [1]

Abstract

Deep learning has shown that learned functions can dramatically outperform hand-designed functions on perceptual tasks. Analogously, this suggests that learned optimizers may similarly outperform current hand-designed optimizers, especially for specific problems. However, learned optimizers are notoriously difficult to train and have yet to demonstrate wall-clock speedups over hand-designed optimizers, and thus are rarely used in practice. Typically, learned optimizers are trained by truncated backpropagation through an unrolled optimization process resulting in gradients that are either strongly biased (for short truncations) or have exploding norm (for long truncations). In this work we propose a training scheme which overcomes both of these difficulties, by dynamically weighting two unbiased gradient estimators for a variational loss on optimizer performance, allowing us to train neural networks to perform optimization of a specific task faster than tuned first-order methods. We demonstrate these results on problems where our learned optimizer trains convolutional networks faster in wall-clock time compared to tuned first-order methods and with an improvement in test loss.

1. Introduction

Gradient based optimization is a cornerstone of modern machine learning. A large body of research has been targeted at developing improved gradient based optimizers. In practice, this typically involves analysis and development of hand-designed optimization algorithms (Nesterov, 1983; Duchi et al., 2011; Tieleman & Hinton, 2012; Kingma & Ba, 2014). These algorithms generally work well on a wide variety of tasks, and are tuned to specific problems via hyperparameter search. On the other hand, a complementary approach is to *learn* the optimization algorithm (Bengio et al., 1990; Schmidhuber, 1995; Hochreiter et al., 2001; Andrychowicz et al., 2016; Wichrowska et al., 2017; Li & Malik, 2017b; Lv et al., 2017; Bello et al., 2017). That is, to learn a function that performs optimization, targeted at particular problems of interest. In this way, the algorithm may learn *task specific* structure, enabling dramatic performance improvements over more general optimizers.

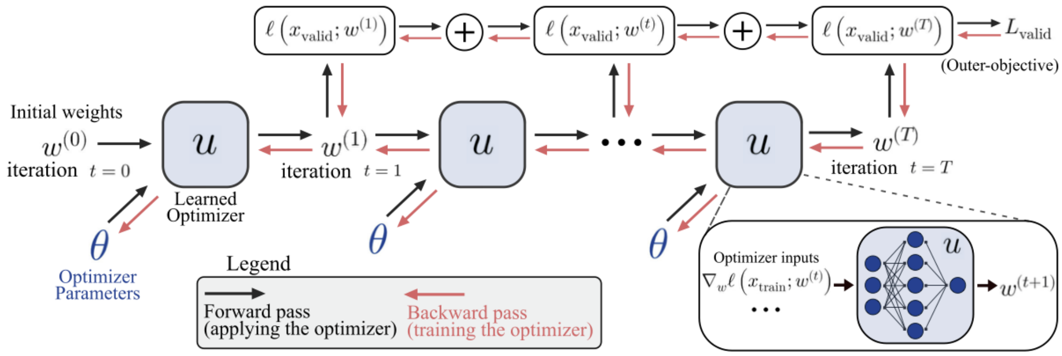
However, training learned optimizers is notoriously difficult. Existing work in this vein can be classified into two broad categories. On one hand are black-box methods such as evolutionary algorithms (Goldberg & Holland, 1988; Bengio et al., 1992), random search (Bergstra & Bengio, 2012), reinforcement learning (Bello et al., 2017; Li & Malik, 2017a;b), or Bayesian optimization (Snoek et al., 2012). However, these methods scale poorly with the number of optimizer parameters.

The other approach is to use first-order methods, by computing the gradient of some measure of optimizer effectiveness with respect to the optimizer parameters. Computing these gradients is costly as we need to iteratively apply the learned update rule, and then backpropagate through these applications, a technique commonly referred to as "unrolled optimization" (Bengio, 2000; Maclaurin et al., 2015). To address the problem of backpropagation through many optimization steps (analogous to many timesteps in recurrent neural networks), many works make use of truncated backpropagation though time (TBPTT) to partition the long unrolled computational graph into separate pieces (Werbos, 1990; Domke, 2012; Tallec & Ollivier, 2017). This not only yields computational savings, at the cost of increased bias (Tallec & Ollivier, 2017), but also limits exploding gradients which emerge from too many iterated non-linear function applications (Pascanu et al., 2013; Parmas et al., 2018). Existing methods have been proposed to address the bias of TBPTT but come at the cost of increased variance or computational complexity (Williams & Zipser, 1989; Ollivier et al., 2015; Tallec & Ollivier, 2017). Previous techniques for training RNNs via TBPTT have thus far not been effective for training optimizers.

In this paper, we analytically and experimentally explore

[1]Google Brain. Correspondence to: Luke Metz <lmetz@google.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.



Term	Definition
$\mathcal{D}$	Dataset consisting of train and validation split, $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{valid}}$.
$\mathcal{T}$	The set of tasks, where each task is a dataset (e.g., a subset of Imagenet classes).
$w^{(t)}$	Parameters of inner-problem at iteration t. These are updated by the learned optimizer, and depend implicitly on θ and $\mathcal{D}_{\text{train}}$.
$\ell\left(x; w^{(t)}\right)$	Loss on inner-problem, for mini-batch x.
θ	Parameters of the optimizer.
$u\left(\cdot;\theta\right)$	Function defining the learned optimizer. The inner-loop update is $w^{(t+1)} = u\left(w^{(t)}, x, \nabla_w \ell, \ldots; \theta\right)$, for $x \sim \mathcal{D}_{\text{train}}$.
$L_{\text{train}}\left(\theta\right)$	Outer-level objective targeting training loss, $\mathbb{E}_{\mathcal{D}\sim\mathcal{T}}\mathbb{E}_{x\sim\mathcal{D}_{\text{train}}}\left[\frac{1}{T}\sum_{t=1}^{T}\ell\left(x; w^{(t)}\right)\right]$.
$L_{\text{valid}}\left(\theta\right)$	Outer-level objective targeting validation loss, $\mathbb{E}_{\mathcal{D}\sim\mathcal{T}}\mathbb{E}_{x\sim\mathcal{D}_{\text{valid}}}\left[\frac{1}{T}\sum_{t=1}^{T}\ell\left(x; w^{(t)}\right)\right]$.
$\mathcal{L}\left(\theta\right)$	The variational (smoothed) outer-loop objective, $\mathbb{E}_{\tilde{\theta}\sim\mathcal{N}(\theta,\sigma^2 I)}\left[L\left(\tilde{\theta}\right)\right]$.

Figure 1. **Top:** Schematic of unrolled optimization. **Bottom:** Definition of terms used in this paper.

the debilitating role of bias and exploding gradients on training optimizers (§2.3). We then show how these pathologies can be remedied by optimizing the parameters of a distribution over the optimizer parameters, known as variational optimization (Staines & Barber, 2012) (§3). We define two unbiased gradient estimators for this objective: a reparameterization based gradient (Kingma & Welling, 2013), and evolutionary strategies (Rechenberg, 1973; Nesterov & Spokoiny, 2011). By dynamically reweighting the contribution of these two gradient estimators (Fleiss, 1993; Parmas et al., 2018; Buckman et al., 2018), we are able to avoid exploding gradients and unroll longer, to stably and efficiently train learned optimizers.

We demonstrate the utility of this approach by training a learned optimizer to target optimization of small convolutional networks on image classification (§4). With our method, we are able to outer-train on more inner steps (10k inner-parameter updates) with more complex inner-problems than prior work. Additionally, we can simplify the parametric form of the optimizer, utilizing a small MLP without any complex tricks such as extensive use of normalization(Metz et al., 2018), or annealing training from existing algorithms(Houthooft et al., 2018) previously needed for stability.

On the targeted task distribution, this learned optimizer achieves better test loss, and is faster in *wall-clock time*, compared to hand-designed optimizers such as SGD+Momentum, RMSProp, and ADAM (Figure 6). To our knowledge, this is the first instance of a learned optimizer performing comparably to existing methods on wall-clock time, as well as the first parametric optimizer outer-trained against validation loss. While not the focus of this work, we also find that the learned optimizer demonstrates promising generalization ability on out of distribution tasks (Figure 7).

2. Unrolled optimization for learning optimizers

2.1. Problem Framework

Our goal is to learn an optimizer which is well suited to some set of target optimization tasks. Throughout the paper, we will use the notation defined in Figure 1. Learning an optimizer can be thought of as a bi-level optimization problem (Franceschi et al., 2018), with *inner* and *outer* levels. The inner minimization consists of optimizing the weights (w) of a target problem $\ell(w)$ by the repeated application of an update rule ($u\left(\cdot\right)$). The update rule is a parameterized function that defines how to map the weights at iteration t to iteration $t+1$: $w^{(t+1)} = u(w^{(t)}, x, \nabla_w \ell, ...; \theta)$. Here, θ represents the parameters of the learned optimizer. In the outer loop, these optimizer parameters (θ) are updated so as to minimize some measure of optimizer performance, the outer-objective ($L(\theta)$). Our choice for L will be the average value of the target loss ($\ell(\cdot)$) measured over either training or validation data. Throughout the paper, we use *inner-* and *outer-* prefixes to make it clear when we are referring to applying a learned optimizer on a target problem (inner) versus training a learned optimizer (outer).

2.2. Unrolled optimization

In order to train an optimizer, we wish to compute derivatives of the outer-objective L with respect to the optimizer parameters, θ. Doing this requires unrolling the optimization process. That is, we can form an unrolled computational graph that consists of iteratively applying an optimizer (u) to optimize the weights (w) of a target problem (Figure 1). Computing gradients for the optimizer parameters involves backpropagating the outer loss through this unrolled computational graph. This is a costly operation, as the entire inner-optimization problem must be unrolled in order to get a single outer-gradient. Partitioning the unrolled computation into separate segments, known as truncated backpropagation, allows one to compute multiple outer-gradients over shorter segments. That is, rather than compute the full gradient from iteration $t = 0$ to $t = T$, we compute gradients in windows from $t = a$ to $t = a + \tau$. The gradients from these segments can be used to update θ *without* unrolling all T iterations, dramatically decreasing the computation needed for each update to θ. The choice for the number of inner-steps per truncation is challenging. Using a large number of steps per truncation can result in exploding gradients making outer-training difficult, while using a small number of steps can produce biased gradients resulting in poor performance. In the following sections we analyze these two problems.

2.3. Exponential explosion of gradients with increased sequence length

We can illustrate the problem of exploding gradients analytically with a simple example: learning a learning rate. Following the notation in Figure 1, we define the optimizer as:

$$w^{(t+1)} = u(w^{(t)}; \theta) = w^{(t)} - \theta \nabla \ell \left(w^{(t)} \right),$$

where θ is a scalar learning rate that we wish to learn for minimizing some target problem $\ell(w^{(t)})$. For simplicity, we assume a deterministic loss ($\ell(\cdot)$) with no batch of data (x).

The quantity we are interested in is the derivative of the loss after T steps of gradient descent with respect to θ. We can compute this gradient (see Appendix A) as:

$$\frac{d\ell(w^{(T)})}{d\theta} = \left\langle g^{(T)}, -\sum_{i=0}^{T-1} \left(\prod_{j=i+1}^{T-1} (I - \theta H^{(j)}) \right) g^{(i)} \right\rangle,$$

where $g^{(i)}$ and $H^{(j)}$ are the gradient and Hessian of the target problem $\ell(w)$ at iteration i and j, respectively. We see that this equation involves a sum of products of Hessians. In particular, the first term in the sum involves a product over the *entire sequence* of Hessians observed during training. When optimizing a quadratic loss, the Hessian is constant, and the outer-gradient becomes a matrix polynomial of degree T, where T is the number of gradient descent steps. Thus, the outer-gradient can grow exponentially with T if the maximum eigenvalue of $(I - \theta H^{(j)})$ is greater than 1. In general, the Hessian is not a constant, but in practice we still see an exponential growth in gradient norm across a variety of settings.

We can see another problem with long unrolled gradients empirically. Consider the task of optimizing a loss surface with two local minima defined as $\ell(w) = (w-4)(w-3)w^2$ with initial condition $w^{(0)} = -1.2$ using a momentum based optimizer with a parameterized momentum value θ (Figure 2a). At low momentum values the optimizer converges in the first of the two local minima, whereas for larger momentum values the optimizer settles in the second minimum. With even larger values of momentum, the iterate oscillates between the two minima before settling. We visualize both the trajectory of $w^{(t)}$ over training and the final loss value for different momentum values in Figure 2b and 2c. With increasing unrolling steps, the loss surface as a function of the momentum θ becomes less and less smooth, and develops near-discontinuities at some values of the momentum resulting in extremely large gradient norms. This behavior is not unique to momentum optimizer parameters. In Appendix B we perform additional experiments modifying learning rates instead and show similar behavior.

Although these are toy systems, we see similar pathologies when optimizing more complex inner-models with both hand designed and learned optimizers. Trajectories taken during optimization often change dramatically as a result of only small changes in optimizer parameters. To illustrate this, in Figure 2d we train a 2 layer MLP with ReLU activations for 40 iterations using Adam. We vary the learning rate between 0.1469 to 0.1484 with 100 samples spaced uniformly in log scale, but keep all sources of randomness fixed. We plot 2D random projections of the MLP's parameters during training, using color to denote different learning rates. Early in training the trajectories are similar, but as more steps are taken the trajectories diverge, producing drastically different trained models with only small

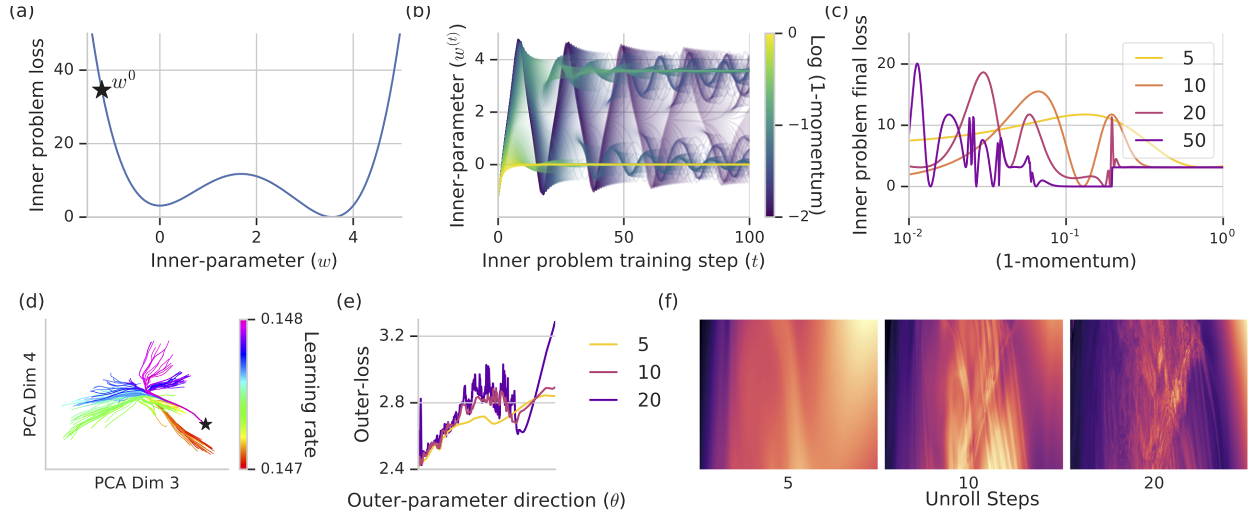


Figure 2. Outer-problem optimization landscapes can become increasingly pathological with increasing inner-problem step count. **(a)** A toy 1D inner-problem loss surface with two local minimum. Initial parameter value ($w^{(0)}$) is indicated by the star. **(b)** Final inner-parameter value ($w^{(T)}$) as a function of the number of inner problem training steps T, when inner-problem training is performed by SGD+momentum. Color denotes different values of the momentum parameter. Low momentum (yellow) converge to the first local minimum at $w = 0$. Slightly higher momentum (green) escape this minimum to settle at the global minimum ($w \approx 3.5$). Even larger values (purple) oscillate before eventually settling in one of the two minima. **(c)** The final loss after some number steps of optimization (shown in different colors) as a function of the momentum. The final loss surface is smooth for small number of training steps T. However, larger values of T result in near discontinuous loss surfaces around the transition points between the two minima. **(d)** Random 2D projection of the parameters of a small MLP trained with Adam using different learning rates. As more steps are taken, the inner-training trajectories diverge. **(e)** Similar to **c**, where the inner problem is a two layer MLP, the learned optimizer is the one used in this paper, and for a 1D slice through the outer-parameters θ along the gradient direction. The outer-objective shown is average validation loss for 5, 10, and 20 inner-unrolls. **(f)** 2D rather than 1D slices through θ, for different numbers of inner-loop steps. Intensity indicates value of $L_{\text{train}}(\theta)$; darker is lower. Similar pathologies are observed to those which manifest in the toy problem.

changes in the learning rate.

In the case of both neural network inner-problems and neural network optimizers, the outer-loss surface can grow even more complex with increasing number of unrolling steps. We illustrate this in Figure 2e and 2f for slices through the loss landscape $L(\theta)$ of the outer-problem for a neural network optimizer.

2.4. Increasing bias with truncated gradients

Existing work on learned optimizers often avoids exploding gradients (§2.3) by using a short truncation window. Here, we demonstrate the bias short truncation windows can introduce in unrolled optimization. These results are similar to those presented in Wu et al. (2016), except that we utilize *multiple* truncations rather than a single, shortened unroll. First, consider outer-learning the learning rate of Adam when optimizing a small two layer neural network on MNIST (LeCun, 1998). We initialize Adam with a learning rate of 0.001 and outer-train using increasing truncation amounts (Figure 3ab). Adam is used as the outer-optimizer. Other outer-optimizers can be found in I. Despite initializing close to the optimal learning rate, when outer-training with severely truncated backprop the resulting learning rate decreases, increasing the outer-loss. The sum of truncated outer-gradients are anti-correlated with the true outer-gradient.

3. Towards stable training of learned optimizers

To perform outer-optimization of a loss landscape with high frequency structure like that in Figure 2, one might intuitively want to smooth the outer-objective loss surface. To do this, instead of optimizing $L(\theta)$ directly we instead optimize a smoothed outer-loss $\mathcal{L}(\theta)$,

$$\mathcal{L}(\theta) = \mathbb{E}_{\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 I)} \left[L\left(\tilde{\theta}\right) \right],$$

where σ^2 is a fixed variance (set to 0.01 in all experiments) which determines the degree of smoothing. This is the same approach taken in variational optimization (Staines & Barber, 2012). We can construct two different unbiased gradient estimators for $\mathcal{L}(\theta)$: one via the reparameterization trick (Kingma & Welling, 2013); and one via the "log-derivative trick", similarly to what is done in evolutionary

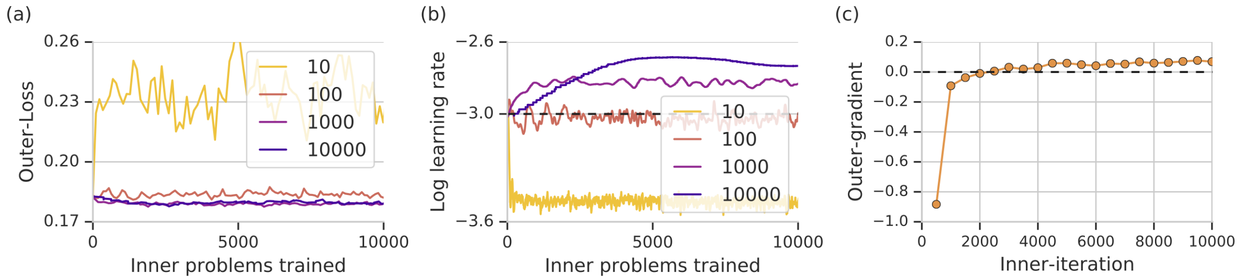


Figure 3. Large biases can result from reducing the number of steps per truncation in unrolled optimization. **(a)** Each line represents an experiment with the same total number of inner-steps (10k) but a different number of unrolling steps per truncation. For each truncation amount, we show an exponential rolling average of outer-loss obtained while outer-training. Short truncations converge to suboptimal outer-loss, while longer truncations (1000, 10000) are less biased and thus achieve lower outer-loss. **(b)** In all cases, the initial learning rate is 0.001 (dashed line). We find low truncation amounts move away from the optimal learning rate, increasing the outer-loss, while larger truncation amounts increase the learning rate, decreasing the outer-loss. **(c)** Outer-gradient at each truncation over the course of inner-training. Initial outer-gradients are highly negative, trying to increase the learning rate, while later outer-gradients are slightly above zero, decreasing learning rate. The total outer-gradient is the sum of these competing contributions.

strategies (ES) and REINFORCE (Williams, 1992; Wierstra et al., 2008). We denote the two estimates as g_{rp} and g_{es} respectively,

$$
\begin{aligned}
g_{\text{rp}} &= \frac{1}{S}\sum_s \nabla_\theta L\left(\theta + \sigma n_s\right), \\
n_s &\sim N\left(0, I\right), \\
g_{\text{es}} &= \frac{1}{S}\sum_s L\left(\tilde{\theta}_s\right) \nabla_\theta \left[\log\left(N\left(\tilde{\theta}_s; \theta, \sigma^2 I\right)\right)\right], \\
\tilde{\theta}_s &\sim N\left(\theta, \sigma^2 I\right),
\end{aligned}
$$

where $N\left(\tilde{\theta}_s; \theta, \sigma^2 I\right)$ is the probability density of the given ES sample, $\tilde{\theta}_s$, S is the sample count, and in implementation the same samples can be reused for g_{rp} and g_{es}.

Following the insight from (Parmas et al., 2018) in the context of reinforcement learning[1], we combine these estimates using inverse variance weighting (Fleiss, 1993),

$$g_{\text{merged}} = \frac{g_{\text{rp}}\sigma_{\text{rp}}^{-2} + g_{\text{es}}\sigma_{\text{es}}^{-2}}{\sigma_{\text{rp}}^{-2} + \sigma_{\text{es}}^{-2}}, \quad (1)$$

where σ_{rp}^2 and σ_{es}^2 are empirical estimates of the variances of g_{rp} and g_{es} respectively. When outer-training learned optimizers we find the variances of g_{es} and g_{rp} can differ by as many as 20 orders of magnitude (Figure 4). This merged estimator addresses this by having at most the lowest of the two variances. To further reduce variance, we employ antithetic sampling. Each normal distribution draw is used twice, both positive and negative, when computing g_{rp} and g_{es}.

[1] (Parmas et al., 2018) go on to propose a more sophisticated gradient estimator that operates on a per iteration level. While this should result in an even lower variance estimator in our setting, we find that the simpler solution of combing both terms at the end is easier to implement and works well in practice.

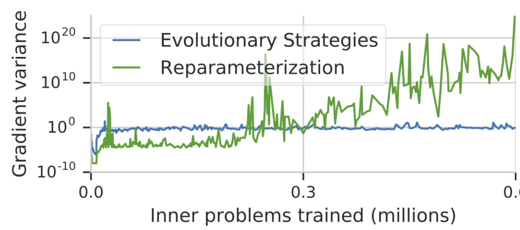


Figure 4. As outer-training progresses, the variance of the reparameterization gradient estimator grows, while the variance of the ES estimator remains constant. The variance of the reparameterization gradient progresses from approximately five orders of magnitude better than that of the ES gradient, to approximately twenty orders of magnitude worse.

The cost of computing a single sample of g_{es} and g_{rp} is thus two forward and two backward passes of an unrolled optimization. To compute the empirical variance, we leverage data parallelism to compute multiple samples of g_{es} and g_{rp}. In theory, to prevent bias the samples used to evaluate σ_{rp}^2 and σ_{es}^2 must be independent of those used to estimate g_{es} and g_{rp}, but in practice we found good performance using the same samples for both.

This gradient estimator fixes the exploding gradients problem when computing gradients over a long truncation, and longer truncations enable lower bias gradient estimates. In practice, these longer truncations are computationally expensive, and early in outer-training shorter truncations are sufficient. The full outer-training algorithm is described in Appendix C.

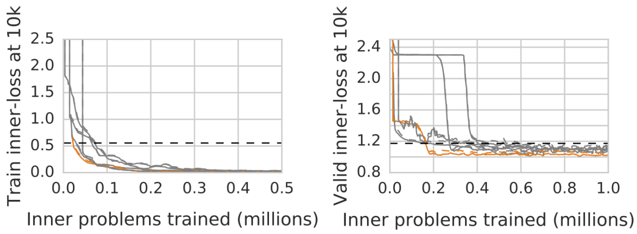


Figure 5. Performance after 10k iterations of inner-training for different learned optimizers over the course of outer-training. Each line represents a different random initialization of outer-parameters (θ). Optimizers are trained targeting the train outer-objective (left), and the validation outer-objective (right). Dashed lines indicate performance of learning rate tuned Adam. Models in orange are the best performing and used in §4.

4. Experiments

As a proof of principle, we use the training algorithm described in §3 to train a simple learned optimizer. For this work, we focus on training an optimizer to target a specific architecture. In the following sections we describe the optimizer architecture used, the task distribution on which we outer-train, as well as outer training details. We then discuss the performance of our learned optimizer on both in and out of distribution target problems. We finish with an ablation study showing the importance of our gradient estimator, as well as aspects of the optimizer's architecture.

4.1. Optimizer architecture

The optimizer architecture used in all experiments consists of a small, fast to compute, fully connected neural network, with one hidden layer containing 32 ReLU units ($\sim$1k parameters). This network is applied to each target problem inner-parameter independently. The outputs of the MLP consist of an un-normalized update direction and a per parameter log learning rate which gets exponentiated. These two quantities are multiplied and subtracted from the previous inner-parameter value to form the next inner-parameter value. The MLP for each weight takes as input: the gradient with respect to that weight, the parameter value, exponentially weighted moving averages of gradients at multiple time scales (Lucas et al., 2018), as well as a representation of the current iteration number. Many of these input features were motivated by (Wichrowska et al., 2017). We conduct ablation studies for these inputs in §4.5. See Appendix D for further architectural details.

4.2. Optimizer target problem

The problem that each learned optimizer is trained against ($\ell(\cdot)$) consists of training a three layer convolutional neural network (32 units per layer, 20k parameters) inner-trained for ten thousand inner-iterations on 32x32x3 image classification tasks. Due to the weight sharing in convolutions, the compute per parameter is high and thus relatively less computation is needed for each inner-iteration. We split the Imagenet dataset (Russakovsky et al., 2015) by class into 700 training and 300 test classes, and sample training and validation problems by sampling 10 classes at random using all images from each class. This experimental design lets the optimizer learn problem specific structure (e.g. convolutional networks trained on object classification), but does not allow the optimizer to memorize class-specific weights for the base problem. This task is modeled after the fact that standard architectures, e.g. ResNet (He et al., 2016), are often applied to a variety of different datasets. See Appendix D for further details.

4.3. Outer-training

To train the optimizer, we linearly increase the number of unrolled steps from 50 to 10,000 over the course of 5,000 outer-training weight updates. The number of unrolled steps is additionally jittered by a small percentage (sampled uniformly up to 20%). Due to the heterogeneous, small iterated computations, we train with asynchronous, batched SGD using 128 CPU workers.

Figure 5 shows the performance of the optimizer (averaged over 40 randomly sampled outer-train and outer-test inner-problems) while outer-training. Despite the stability improvements described in the last section, there is still variability in optimizer performance over random initializations of the optimizer parameters. As expected given our optimizer parameterization, there is very little outer-overfitting. Nevertheless, we use outer-training loss to select the best model and use this in the remainder of the evaluation.

4.4. Learned optimizer performance

Figure 6 shows performance of the learned optimizer, after outer-training, compared against other first-order methods on a sampled validation task (classes not seen during outer-training). For "Adam", "RMSProp", and "Momentum", we report the best performance after tuning the learning rate by grid search using 11 values over a logarithmically spaced range from 10^{-4} to 10 on a per task basis. Searching over fixed learning rates is the baseline most commonly used in other learned optimizer work (Andrychowicz et al., 2016; Wichrowska et al., 2017). However, practitioners often adjust many more hyperparameters. Therefore, we provide an additional baseline consisting of tuning: all Adam optimizer parameters (beta1, beta2, epsilon, learning rate); learning rate decay (exponential decay coefficient, linear decay coefficient); and regularization parameters (l1 regularization, l2 regularization). We outer-

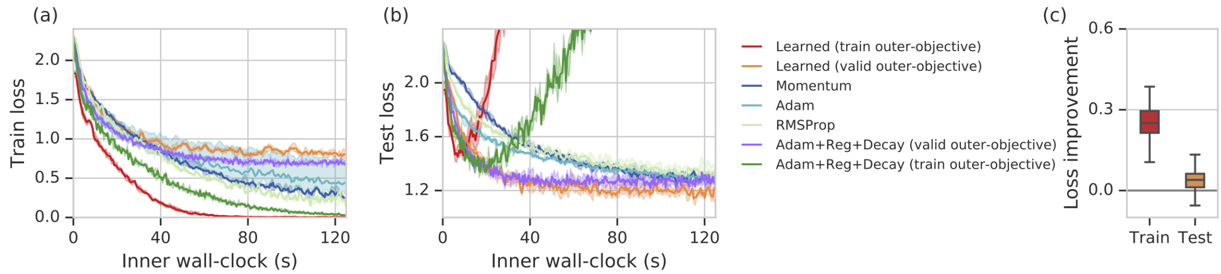


Figure 6. Learned optimizers outperform existing optimizers on training loss (a) and test loss (b). **(a,b)** Training and test curves on outer-test tasks (tasks not seen during outer-training). We show two learned optimizers – one trained to minimize training loss, and the other trained to minimize validation loss on the inner-problem. We provide 2 types of baselines. First, learning rate tuned Adam, RMSProp, and SGD+Momentum, for both train and validation loss (Panel (a) and (b), respectively). Second, we show tuned Adam (learning rate, beta1, beta2, epsilon), with learning rate schedules (linear decay and exponential decay values), and regularization (multipliers on l1 and l2 added to the inner-loss). We tune with random search against both train and validation loss. On training loss (a), our learned optimizer approaches zero training loss faster than existing methods. On test loss (b), our learned optimizer is of similar speed, but converges to a lower minimum. Shaded regions correspond to 25 and 75 percentile over five random initializations of the CNN. For plots showing performance in terms of step count rather than wall-clock, and for more task instances, see Appendix E. See Appendix G for details on the wall-clock calculations. **(c)** Distribution of the performance difference between the learned optimizers and corresponding Adam+Reg+Decay baseline. Positive values indicate performance better than baseline. We show training and test losses for the outer-testing task distribution (tasks not seen during training of the optimizer). On the majority of tasks, the learned optimizers outperform the baseline.

optimize these parameters against the distribution of tasks, evaluating >2k parameter combinations totalling ~100k total inner-problem training runs (performance is averaged over 80 inner runs for a lower variance estimate of L). As with the learned optimizer, we outer-optimize with respect to L_{train}, and L_{test}. We outer-optimize our baseline using uniform random search over all 8 hyperparameters. Outer-optimizing the baseline with the framework we present in this paper yields similar results both in compute cost, and final performance achieved. We performed a similar optimization for RMSProp and SGD+Momentum and find similar performance to that of Adam. Results are not shown here for clarity but can be found in Appendix E.

When outer-trained against the training outer-objective, L_{train}, our learned optimizer achieves faster convergence on training loss (Figure 6a), but poor performance on test loss (Figure 6b). This is expected, as our outer-training procedure never sees validation loss, and thus only minimizes training loss, causing overfiting. When outer-trained against the validation outer-objective, L_{valid}, we also achieve fast optimization and reach a lower test loss in the given time interval (Figure 6b). We suspect further gains in validation performance could be obtained with the addition of more regularization techniques into both the learned optimizer, and the tuned Adam baseline.

Figure 6c summarizes the performance of the learned optimizer across 100 sampled outer-test tasks (tasks not seen during outer-training). It shows the difference in loss (averaged over the first 10k iterations of training) between the learned optimizer and the 8 parameter Adam (Adam+Reg+Decay) baseline tuned against the corresponding loss. Our learned optimizer outperforms this baseline on the majority of tasks.

Although the focus of our approach was *not* generalization, we find that our learned optimizer nonetheless generalizes to varying degrees to dissimilar datasets, different numbers of units per layer, different number of layers, and even to fully connected networks. In Figure 7 we show performance on a six layer convolutional neural network trained on MNIST. Despite the different number of layers, dissimilar dataset, and different input size, the learned optimizers still reduces the loss, and in the case of the validation outer-objective trains faster and generalizes well. We further explore the limits of generalization of our learned optimizer on additional tasks in Appendix F.

4.5. Ablations

To assess the importance of the gradient estimator discussed in §3, the unrolling curriculum §4.3, as well as the features fed to the optimizer enumerated in §4.1, we retrained the learned optimizer removing each of these additions. In particular, we trained optimizers with: only the reparameterization gradient estimator (Gradients), only with evolutionary strategies (ES), a fixed number unrolled steps per truncation (10, 100, 1000) as opposed to a schedule keeping while keeping the same total inner-weight updates, no momentum terms (No Mom), and without the current iteration (No Time). To account for variance, each configuration is repeated with multiple random seeds. Figure 8

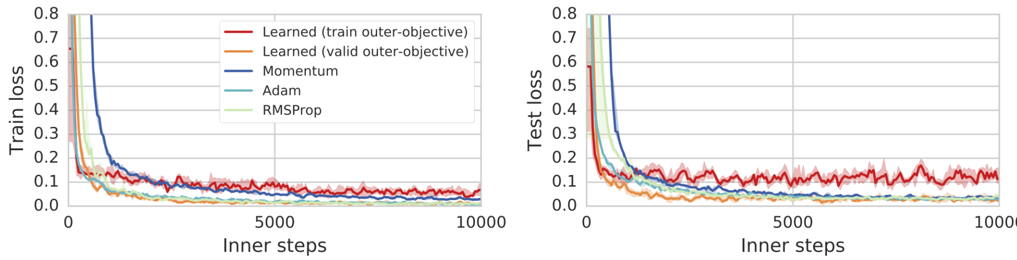


Figure 7. Train and test learning curves for an out of distribution training task consisting of a six layer convolutional neural network trained on 28x28 MNIST. We compare against learning rate tuned Adam, RMSProp, and SGD+Momentum. Note that the optimizer trained to target validation loss generalizes better than the one trained to target train loss. See Appendix F for experiments testing generalization to additional tasks.

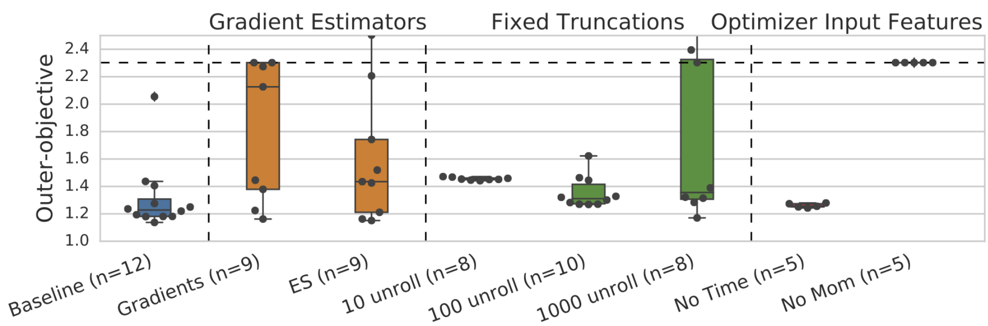


Figure 8. Ablation study. We compare the model described in §4.1 with different features removed. Shown above is the distribution of outer-loss performance across n random seeds averaged between 380k and 420k inner problem trained for each seed. For full learning curves, see Appendix H. We find the combined gradient estimator is more stable and outperforms both the analytic gradient, and evolutionary strategies. To test the role of the truncation length curriculum, we show different, fixed truncation lengths. We find poor convergence with shorter unrolls (10, 100 steps per unroll) and high variance with longer unrolls (1000 steps per unroll). Finally, we find our learned optimizers perform nearly as well *without* the training step as an input, but fails to converge when not given access to momentum based features.

summarizes these findings, showing the learned optimizer performance for each of these ablations. We find that the gradient estimator (in §3) and an increasing schedule of unroll steps are critical to performance, along with including momentum as an input to the optimizer.

5. Discussion

In this work we demonstrate two difficulties when training learned optimizers: "exploding" gradients, and a bias introduced by truncated backpropagation through time. To combat this, we construct a variational bound of the outer-objective and minimize this via a combination of reparameterization and ES style gradient estimators. By using our combined estimator and a curriculum over truncation step we are able to train learned optimizers that are faster in wall-clock time compared to existing optimizers.

In this work, we focused on applying optimizers to a restricted family of tasks. While useful in its own right (e.g. rapid retraining of models on new data), future work will explore the limits of "no free lunch" (Wolpert & Macready, 1997) in the context of optimizers, to understand how and when learned optimizers generalize across tasks. We are also interested in using these methods to better understand what problem structure our learned optimizers exploit. By analyzing the trained optimizer, we hope to develop insights that may transfer back to hand-designed optimizers. Outside of meta-learning, we believe the outer-gradient estimator presented here can be used to train other long time dependence recurrent problems such as neural turning machines (Graves et al., 2014), or neural GPUs (Kaiser & Sutskever, 2015).

Much in the same way deep learning has replaced feature design for perceptual tasks, we see meta-learning as a tool capable of learning new and interesting algorithms, especially for domains with unexploited problem-specific structure. With better outer-training stability, we hope to improve our ability to learn interesting algorithms, both for optimizers and beyond.

ACKNOWLEDGMENTS

We would like to thank Madhu Advani, Alex Alemi, Samy Bengio, Brian Cheung, Chelsea Finn, Sam Greydanus, Hugo Larochelle, Ben Poole, George Tucker, and Olga Wichrowska, as well as the rest of the Brain Team for conversations that helped shape this work.

References

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

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Bello, I., Zoph, B., Vasudevan, V., and Le, Q. Neural optimizer search with reinforcement learning. 2017. URL https://arxiv.org/pdf/1709.07417.pdf.

Bengio, S., Bengio, Y., Cloutier, J., and Gecsei, J. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pp. 6–8. Univ. of Texas, 1992.

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

Bengio, Y., Bengio, S., and Cloutier, J. *Learning a synaptic learning rule*. Université de Montréal, Département d'informatique et de recherche opérationnelle, 1990.

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

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. Sample-efficient reinforcement learning with stochastic ensemble value expansion. *arXiv preprint arXiv:1807.01675*, 2018.

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

Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

Fleiss, J. Review papers: The statistical basis of meta-analysis. *Statistical methods in medical research*, 2(2): 121–145, 1993.

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

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

Goldberg, D. E. and Holland, J. H. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

Graves, A., Wayne, G., and Danihelka, I. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Houthooft, R., Chen, R. Y., Isola, P., Stadie, B. C., Wolski, F., Ho, J., and Abbeel, P. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

Kaiser, Ł. and Sutskever, I. Neural gpus learn algorithms. *arXiv preprint arXiv:1511.08228*, 2015.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

LeCun, Y. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

Li, K. and Malik, J. Learning to optimize. *International Conference on Learning Representations*, 2017a.

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

Lucas, J., Zemel, R., and Grosse, R. Aggregated momentum: Stability through passive damping. *arXiv preprint arXiv:1804.00325*, 2018.

Lv, K., Jiang, S., and Li, J. Learning gradient descent: Better generalization and longer horizons. *arXiv preprint arXiv:1703.03633*, 2017.

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

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

Nesterov, Y. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pp. 543–547, 1983.

Nesterov, Y. and Spokoiny, V. Random gradient-free minimization of convex functions. Technical report, Université catholique de Louvain, Center for Operations Research and Econometrics (CORE), 2011.

Ollivier, Y., Tallec, C., and Charpiat, G. Training recurrent networks online without backtracking. *arXiv preprint arXiv:1507.07680*, 2015.

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.

Rechenberg, I. Evolutionsstrategie–optimierung technisher systeme nach prinzipien der biologischen evolution. 1973.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Schmidhuber, J. On learning how to learn learning strategies. 1995.

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.

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

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*, 4(2):26–31, 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. *International Conference on Machine Learning*, 2017.

Wierstra, D., Schaul, T., Peters, J., and Schmidhuber, J. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pp. 3381–3387. IEEE, 2008.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

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

Wolpert, D. H. and Macready, W. G. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

Wu, Y., Ren, M., Liao, R., and Grosse, R. B. Understanding short-horizon bias in stochastic meta-optimization. pp. 478–487, 2016.