A Simple Guard for Learned Optimizers

Isabeau Prémont-Schwarz [* 1] **Jaroslav Vítků** [* 1] **Jan Feyereisl** [1]

Abstract

If the trend of learned components eventually
outperforming their hand-crafted version contin-
ues, learned optimizers will eventually outper-
form hand-crafted optimizers like SGD or Adam.
Even if learned optimizers (L2Os) eventually out-
pace hand-crafted ones in practice however, they
are still not provably convergent and might fail out
of distribution. These are the questions addressed
here. Currently, learned optimizers frequently out-
perform generic hand-crafted optimizers (such as
gradient descent) at the beginning of learning but
they generally plateau after some time while the
generic algorithms continue to make progress and
often overtake the learned algorithm as Aesop's
tortoise which overtakes the hare. L2Os also still
have a difficult time generalizing out of distribu-
tion. ([Heaton et al., 2020]) proposed Safeguarded
L2O (GL2O) which can take a learned optimizer
and safeguard it with a generic learning algorithm
so that by conditionally switching between the
two, the resulting algorithm is provably conver-
gent. We propose a new class of Safeguarded
L2O, called Loss-Guarded L2O (LGL2O), which
is both conceptually simpler and computationally
less expensive. The guarding mechanism decides
solely based on the expected future loss value of
both optimizers. Furthermore, we show theoreti-
cal proof of LGL2O's convergence guarantee and
empirical results comparing to GL2O and other
baselines showing that it combines the best of
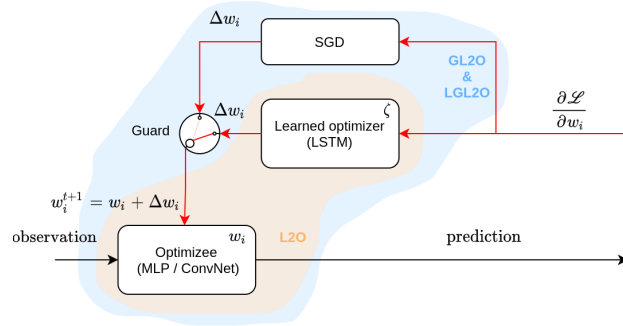both L2O and SGD and that in practice converges
much better than GL2O.

Figure 1. Principle of the guarding mechanism for learned optimiz-
ers. A learned optimizer is used to change weights of an optimizee.
Guarded learned optimizers (GL2O and LGL2O) add a guard
which monitors situations where the optimizer does not perform
well and switches to an analytic method (such as SGD) in those
cases. This ensures asymptotic convergence of the resulting hybrid
optimizer.

1. Introduction

An unambiguous trend in machine learning is that different
parts of the pipeline are being automated. Automated
architecture search is now the state of the art for neural
networks ([Tan & Le, 2021]), learned dynamic learning rates
work better than learning rate schedules ([Xu et al., 2019]),
even reinforcement learning algorithms have been learned
([Co-Reyes et al., 2021]; [Alet* et al., 2020]; [Kirsch et al.,
2020]). There has also been quite some work on learned
optimizers (L2O) (cf. section [2]) though so far, due to com-
putational limits, they work only on small datasets and only
beat hand-crafted optimizers for the first thousand steps
or so (because that is the horizon they are trained on) and
do not generalize well. Some, like Richard Sutton([Sutton,
2019]), argue that as compute becomes cheaper and more
accessible, "learned things" will become much better than
"handcrafted things". However, even if L2O can outperform
designed optimizers, they will still have two flaws: they will
not be provably convergent, and they might fail totally out
of distribution (different dataset, different type of objects
being optimized – what we call optimizees in this paper,
different learning horizons, etc.). A guard addresses those
shortcomings.

---
[*]Equal contribution [1]Good AI, Prague, Czechia.
Correspondence to: Isabeau Prémont-Schwarz
<premont-schwarz@goodai.com>, Jaroslav Vítků
<jaroslav.vitku@goodai.com>.

Our main contribution is a guard which takes in as input any blackbox L2O and a provably convergent optimizer and blends them to get the best of both. We prove that our guard keeps the convergence guarantee of the designed optimizer. Then we show in practice that our guard has the desired behaviour, that is, it uses the L2O when the L2O works best (in distribution or in settings where the L2O generalizes well) but correctly switches to the designed optimizer when the L2O underperforms.

In our experiments, we use (Andrychowicz et al., 2016)'s L2O that we train such that it beats other optimizers in the first ∼1000 steps, but our contribution, the guard, is independent of the L2O, it can take any L2O as input, and as L2Os get better so will LGL2O. Thus the goal of the paper is not to show that the combination is better than any other existing optimizer, but rather to show that the guard makes the right decisions and that it preserves the advantages of both its input L2O and provably convergent designed optimizer. We show this in our experiments by demonstrating that LGL2O performs as well as L2O (or better) when L2O outperforms SGD, and as well as SGD (or better) when SGD outperforms L2O. All unguarded L2O approaches currently work only for ∼1000 optimization steps in practice when optimizing neural networks. In contrast we show successful optimization up to millions of steps.

The main reasons we developed a new class of guards, while an existing class of guards (Heaton et al., 2020) already exist are:

- Our guard is conceptually simpler.
- Our guard requires fewer hyperparameters.
- Our guard requires fewer SGD calls, and those can be done in parallel rather than sequentially.
- In practice our guard converges better for neural networks.

The first three points are detailed in section 3 while the last point is detailed in section 4.

2. Related Work

Learning to Optimize (L2O), popularized by Andrychowicz et al. (2016), focuses on learning optimization rules using an LSTM network, where gradients, along with other information, are provided on the input to the learned optimizer and updates to the weights of a base network are provided as outputs. Since then, many iterations of similar approaches with various alterations have been proposed (Metz et al., 2020; Lv et al., 2017; Chen et al., 2021), yet ultimately with the same purpose, i.e. augmenting gradient descent for some practical benefit, such as greater sample efficiency. Historically, however, other works have proposed to use a neural network to train another neural network (Prokhorov et al., 2002; Hochreiter et al., 2001). For a recent overview

of this area, one can refer to Chen et al. (2021).

Recently, limitations of L2O approaches have increasingly been analyzed (Metz et al., 2020; Wu et al., 2018; Metz et al., 2021; Maheswaranathan et al., 2020), revealing a plethora of limitations that hinder the use of L2O methods in practice. Most recently, Heaton et al. (2020) proposed a method for safeguarding the behaviour of any learned optimizer by combining it with stochastic gradient descent (SGD) to confer the hybrid algorithm with convergence guarantees. Our work provides a computationally and conceptually simpler safeguarding mechanism to guarantee convergence.

Learning to optimize approaches belong, more broadly, to the field of meta-learning and learning to learn in particular (Hospedales et al., 2021). Approaches such as the Badger framework (Rosa et al., 2019), VS-ML (Kirsch & Schmidhuber, 2020) and BLUR (Sandler et al., 2021) focus on generalizing learning algorithms and architectures and how to discover and train them. The method proposed in this work deals with challenges that must first be overcome to build learning systems that could one day rival and more importantly surpass existing hand engineered solutions, one of the primary motivators for this work. The existing challenges encompass a wide variety of sub-problems. Amongst others, these include a) the ability of the learned algorithm to generalize outside of meta-training distribution and b) the stable long-term (asymptotic) behaviour of the algorithm. These two topics are the subject of this paper.

3. Loss Guarding

The principle of guarding is illustrated in Figure 1. In L2O, the (neural network-based) learned optimizer receives one gradient per parameter of the optimizee (in what follows we call **optimizee** whatever gets optimized by the optimizer, in our case it is typically a neural network) and independently proposes a corresponding delta for each parameter that is then applied. In case of guarding (GL2O and LGL2O), the guard looks at parameters proposed by L2O and decides whether to accept them or those suggested by traditional convergence-guaranteed optimizer instead (the fallback update). The guarding mechanism uses the guarding criteria to choose whether to use updates proposed by L2O or the updates proposed by the convergence-guaranteed optimizer (eg. SGD). At every step if the L2O updates are chosen then the L2O weight updates are applied to all the weights, and if the convergence-guaranteed optimizer weight updates are chosen, then those updates are applied to all the weights.

The difference between LGL2O and GL2O is the criterion to decide whether to apply the L2O update or the convergence-guaranteed optimizer update. In the case of GL2O, motivated by the convergence of Cauchy sequences to fixed points in a complete space, the update proposed by L2O

is tested by applying normal SGD to it. If the L2O update makes the SGD step size generally smaller (smaller than a Cauchy sequence which tracks the size of accepted steps), then the L2O update is accepted, if not it is rejected and an SGD update is used instead. In contrast, LGL2O, motivated by the idea that convergence in the loss-space implies convergence in the weight-space when the loss function is continuous (and locally convex), simply compares the loss of the L2O update versus the loss of a convergence-guaranteed optimizer like SGD and simply implements the update with the lowest loss. That is at optimization step k, if y_k is the point proposed by L2O and z_k is the point proposed by the convergence-guaranteed generic optimizer, then the criterion to determine the update is

$$\mathcal{F}(y_k) < \mathcal{F}(z_k), \tag{1}$$

where $\mathcal{F}$ is the loss function. If the criterion is passed, the algorithm updates to y_k otherwise z_k.

This means that for every weight update of the optimizee, GL2O needs to make one extra call of obtaining the gradients of the optimizee (on the proposal of L2O) and it must be made sequentially, after making the initial one and running L2O. This significantly increases the time complexity of GL2O compared to LGL2O. The logic of GL2O is also thus more complicated.

Algorithm 1 Loss Guarded L2O with (deterministic) gradient descent

---
1: **Given** task loss function: $\mathcal{F}$
2: **Given** L2O operator: $\mathcal{L}_{L2O}$
3: **Given** L2O weights $\{\zeta\}$ ◁ *Take from Meta-Training*
4: **Given** initial state $x^1 \in \mathbb{R}^n$ ◁ *Initialize iterate*
5: **for** $k = 1, 2, \ldots$ **do**
6:    $y^k \leftarrow \mathcal{L}_{L2O}(x^k; \zeta)$ ◁ *L2O Update*
7:    $z^k \leftarrow x^k - \lambda \nabla \mathcal{F}(x^k)$ ◁ *Fallback Update*
8:    **if** $\mathcal{F}(y^k) < \mathcal{F}(z^k)$ **then** ◁ *Safeguard Check*
9:      $x^{k+1} \leftarrow y^k$ ◁ *L2O Update*
10:   **else**
11:      $x^{k+1} \leftarrow z^k$ ◁ *Fallback Optimizer Update*
12:   **end if**
13: **end for**

In algorithm 2, n_t is number of sequential application of L2O (and in parallel, on the same n_t mini-batches, SGD) before choosing whether to use L2O updates or the fallback SGD updates using criterion 1. n_c is the number of mini-batches (drawn from the training data) used to approximate the loss in criterion 1. In practice we want to choose $n_c \simeq n_t$ for algorithmic speed so that not too many loss function evaluations are needed per optimization step. If we choose $n_c = n_t$, then we need only two loss function evaluation per optimization step while still being able to approximate the total loss function in criterion 1 with an arbitrary number n_c

---

Algorithm 2 Loss Guarded L2O with stochastic gradient descent

---
1: **Given** mini-batch loss function: $\mathcal{F}$
2: **Given** L2O operator: $\mathcal{L}_{L2O}$
3: **Given** L2O weights $\{\zeta^k\}$ ◁ *Take from Meta-Training*
4: **Hyperparameters:** $n_t, n_c, L \in \mathbb{N}^3$
5: **Given** initial state $x^1 \in \mathbb{R}^n$ ◁ *Initialize iterate*
6: $k \leftarrow 1$
7: **while** $k < L$ **do**
8:   Sample n_t train mini-batches $\mathcal{B}_t = [b_1, \ldots, b_{n_t}]$
9:   Sample n_c validation mini-batches $\mathcal{B}_v = [v_1, \ldots, v_{n_c}]$
10:   $y^k \leftarrow x^k$ ◁ *L2O Init.*
11:   $z^k \leftarrow x^k$ ◁ *SGD Init.*
12:   **for** $i \in [0, \ldots, n_t - 1]$ **do**
13:     $y^{k+i+1} \leftarrow \mathcal{L}_{L2O}(y^{k+i}, b_{i+1}; \zeta)$ ◁ *L2O Update*
14:     $z^{k+i+1} \leftarrow z^{k+i} - \lambda_{k+i} \nabla \mathcal{F}(z^{k+i}, b_{i+1})$ ◁ *Fallback Update*
15:   **end for**
16:   **if** $\frac{1}{n_c} \sum_{j=1}^{n_c} \mathcal{F}(y^{k+n_t}, v_j) < \frac{1}{n_c} \sum_{j=1}^{n_c} \mathcal{F}(z^{k+n_t}, v_j)$ **then** ◁ *Safeguard Check*
17:     $x^{k+n_t} \leftarrow y^{k+n_t}$ ◁ *L2O Update*
18:   **else**
19:     $x^{k+n_t} \leftarrow z^{k+n_t}$ ◁ *Fallback Optimizer Update*
20:   **end if**
21:   $k \leftarrow k + n_t$
22: **end while**

---

of mini-batches. In all our experiments, both n_t and n_c are chosen to be 10. These are the only two hyperparameters of the algorithm 2 (except for L, which is how long one chooses to optimize for). This compare favourably with GL2O which has the choice of 5 possible sequence types, each choice then with typically 2 hyperparameters to tune.

Theorem 1. *Let $\mathcal{F}$ be a continuous loss function which is μ-strongly convex[a] and L-smooth and let w^* be it's global minimum. Let $w_{i \in \mathbb{N}}$ be a sequence of points obtained from applying the Loss-Guarded L2O algorithm with gradient descent or stochastic gradient as the guarding algorithm. In the case of stochastic gradient descent, we assume that in expectation, the stochastic gradient $\nabla_{mb}\mathcal{F}(w)$ is equal to the true gradient,*

$$\mathbb{E}(\nabla_{mb}\mathcal{F}(w)) = \nabla\mathcal{F}(w),$$

and that the variance of the stochastic gradient around the true gradient is bounded. Then given a constant learning rate $0 < \lambda < \min(\frac{2}{L}, 2\mu)$ for gradient descent or a decaying learning rate $\lambda_i \propto \frac{1}{i_0 + i}$ for SGD,

the sequence converges to the minimum, i.e.

$$\lim_{i \to \infty} w_i = w^*.$$

---

[a]We remark that any convex function can be turned into a strongly convex one simply by adding an L_2 regularization. In the non-convex case, we get convergence to a local minimum instead of the global one.

Proof. This is simply a combination of Theorems 5 and 7 proven in Appendix. □

Because the LGL2O criterion depends on comparing the proposed points coming from the L2O and SGD on the whole loss function, but in all our experiments the loss function is the sum of the individual losses on many dataset points which would take considerable compute to evaluate, in practice we approximated the loss function in the criterion with 10 mini-batches of data. There is a risk in making this approximation: if the error on the loss from approximating it with a limited number of mini-batches becomes similar to the difference in the loss values of the SGD and L2O update proposals, then there is a risk that the algorithm will choose the incorrect update and convergence will not be guaranteed. This is something which risks happening near a local optimum, in which case one should increase the number of mini-batches used in the evaluation of the loss criterion to lower the approximation error.
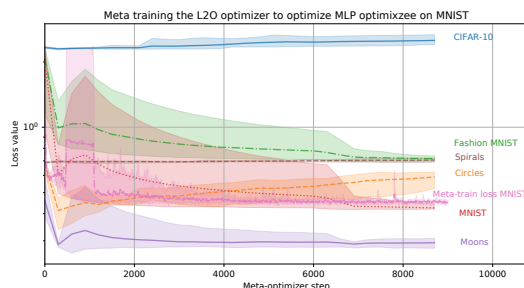
4. Experiments

Figure 2. Outer-loop convergence of the learned optimizer on the MNIST dataset. The L2O method achieves reasonable performance on the MNIST dataset. It generalizes to FashionMNIST & Moons & Circles datasets, while the MLP seems too small to have sufficient capacity to solve the Spirals & CIFAR-10. Mean and min-max ranges are across 5 runs.

This chapter compares the proposed LGL2O with the original GL2O, non-guarded L2O and baseline handcrafted algorithms. We first compare them in distribution, that is when L2O is used like it was meta-trained, on the same dataset, and with the same optimizee. In all our experiments, we

use an L2O that was meta-trained on MNIST on a small fully-connected MLP for roughly 10k meta-optimizer steps. Then we follow with out of distribution experiments. First with only the dataset being out of distribution (other than MNIST), then with only the optimizee being out of distribution (ConvNets instead of MLPs), and finally both.

To show that the success of our guard is not dependent on the above specific L2O, we also meta-trained another L2O in a different context (Convnet on CIFAR-10) and show in appendix D that our guard works just as well with this other learned optimizer as with the MLP-MNIST-metatrained L2O we used in this section.

The experiments were conducted on publicly available datasets, namely MNIST (LeCun & Cortes, 2010), Fashion-MNIST (Xiao et al., 2017), CIFAR10 (Krizhevsky, 2009), TinyImagenet[1] - a subset of Imagenet dataset (Russakovsky et al., 2015) and simple datasets from the Scikit-learn library (Pedregosa et al., 2011).

The purpose of these experiments is to illustrate a common property of meta-learned algorithms: that they tend to converge faster than analytic algorithms and then they plateau. It is expected that the guarding mechanism in GL2O will eventually take over and switch to the SGD-based guard, thus assuring the asymptotic convergence towards the optimal solution.

The learned optimizer (and it's weights) is identical in all experiments and consists of an LSTM (Hochreiter & Jurgen Schmidhuber, 1997) with 2 hidden layers of 20 cells each and a linear output layer which was meta-trained with a rollout-length of 100 steps to optimize an MLP on the MNIST dataset. Pre-processing of the input is used as described in (Andrychowicz et al., 2016).

First, the optimizer was meta-trained to optimize the optimizee on the MNIST dataset as shown in the Figure 2. The optimizee used for meta-training is an MLP with 1 hidden layer with 20 neurons. It uses sigmoid activations in hidden layer(s), softmax on the output layer and a negative log-likelihood loss.

After the meta-learning phase, the meta-testing phase begins. At that point, the learned optimizer works in an inference regime. In all these experiments we explore the behaviour of LGL2O ("LGL2O (ours)") in the plots) is compared with Guarded L2O from Heaton et al. (2020) (GL2O in the plots), vanilla L2O (L2O in the plots) and SGD without momentum (SGDnm) (which the fallback optimizer which we use inside LGL2O). In addition, each figure plots "LGL2O use_l2o" which tracks whether the L2O update (use_l2o=1) or the SGD update (use_l2o=0.5) was chosen by the loss-guarding

---

[1]TinyImagenet dataset is publicly available from Kaggle competition website at https://www.kaggle.com/c/tiny-imagenet/data

In Distribution?		Experiment Type	
Dataset	**Optimizee**	**Dataset**	**Optimizee**
yes	yes	MNIST	MLP
yes	no	MNIST	Conv
no	yes	Spirals & Circles	MLP
no	no	CIFAR-10	Conv
no	no	TinyImagenet50	Conv

Table 1. Overview of the experiments: after the meta-training phase, the ability of the hybrid optimizer LGL2O is evaluated on long rollouts, on in-distribution and out-of-distribution datasets and optimizees.

criterion (1) on this optimization step.

The ability to generalize to longer rollouts and to out-of-distribution dataset and network architectures is systematically evaluated on the following experiments, the experiments are described in table 1. In all graphs, the data are collected from 5 independent runs with different seeds. The dark lines trace the mean of the 5 seeds and the shading the minimum and maximum.

In practice (in all experiments shown here) the following schedule for guarding in LGL2O is used:

- make 10 optimizer steps on 10 consecutive mini-batches (for both L2O and guard),

- compute loss of the resulting optimizee as an average of 10 unseen mini-batches (for both optimizees produced by L2O and guard).

This scheme has two motivations. First: a typical learned optimizer behaves differently than SGD, it usually starts with a form of triangulation of the loss landscape, during which the loss increases considerably (see the initial steps in Figure 5) and then the loss starts decreasing. We observed that this stage is necessary for L2O to work. This is the reason why each of the optimizers do 10 consecutive steps first. The second part (evaluating loss over 10 "test" mini-batches) aims to inhibit influence of noise in the evaluation. Without averaging, LGL2O exhibited very unstable behaviour caused by switching from the guard back and forth inappropriately.

4.1. In-distribution dataset and optimizee

This experiment uses the dataset and optimizee that was used during the meta-training phase. Therefore it evaluates the asymptotic behaviour of the learned L2O optimizer. It illustrates the typical flaw of learned optimizers well, which is to say that they perform extremely well at the beggining of optimization but then start diverging.
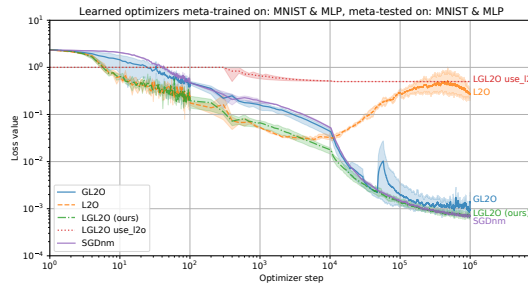

Figure 3. **In-distribution dataset and optimizee**. A typical behaviour of L2O (green) is shown here: at beginning, it converges faster than other algorithms, however, after around 1000 steps it starts diverging. Compared to this, the LGL2O algorithm detects that L2O performs worse than the guard, so it starts using the guard optimizer (the 'use_l2o' line goes towards value 0.5, which corresponds to full use of the guard) and thus stays as good or better than either components (L2O and SGDnm) at all times. As a result, LGL2O combines benefits of both worlds: thanks to the L2O optimizer, it converges quickly at the beginning, while it preserves asymptotic convergence of the guarding algorithm (SGD without momentum here). Note: on all graphs a moving window averaging is used, which starts at Optimizer step (x-axis) 300. The following is common for all the figures below: **LGL2O:** loss of our LGL2O algorithm, **GL2O:** loss of the GL2O algorithm of (Heaton et al., 2020), **L2O:** loss of vanilla L2O, **SGDnm:** loss of SGD without momentum, **LGL2O use_l2o:** indicator function which indicates whether on this step LGL2O used the L2O update (=1) or the SGD update (=0.5)

Figure 3 shows that L2O converges quickly at first, but eventually the loss starts diverging. The LGL2O algorithm solves this by detecting when the handcrafted optimizer would be preferable and switching to it. It can be seen that the switching is not definitive, once L2O performs better than the guard, it can be used again instead of the guard. As expected, the resulting hybrid algorithm steadily converges towards the optimum.

The motivation of guarded learned optimizers is to combine the best of both worlds: to achieve quick convergence at the beginning and then be able to keep converging towards the optimum. LGL2O shows this kind of behaviour in Figure 3.

We also observe that while GL2O prevents the divergence of the L2O, it does at the cost of much of the performance gains of L2O at the beginning of optimization (between steps 0 and 100, where GL2O is better than SGDnm but much worse than vanilla L2O).
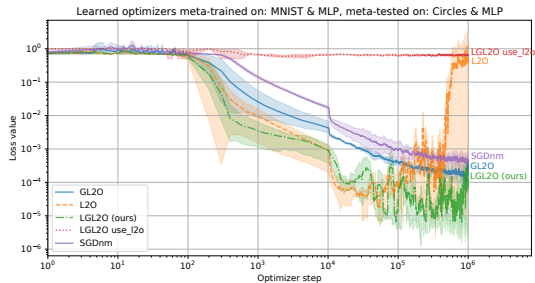
Figure 4. **Out-of-distribution dataset, in-distribution optimizee.** Performance of LGL2O is comparable to L2O at the beginning, where all the optimizers converge quickly. However, L2O starts diverging later in the rollout, while LGL2O correctly switches towards the guard. Again we see that GL2O prevents divergence but at the cost of underperforming compared to L2O at the beginning. We suspect that the reason LGL2O starts diverging after 20k steps is because we evaluate the criterion on only 10 mini-batches, and when the loss is already so low, the variance between different groups of 10 mini-batches becomes larger than the difference between the losses of the SGD proposed update and L2O proposed update, so our implementation of the algorithm starts incorrectly using the divergent L2O too often.
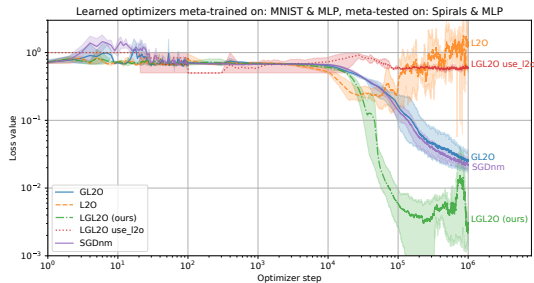

Figure 5. **Out-of-distribution dataset, in-distribution optimizee.** MLP optimizee optimized using the Spirals dataset. The dataset is relatively challenging for MLP with 1 small hidden layer. It can be seen that the SGD baseline without momentum gets stuck at a suboptimal solution forever and Adam gets stuck there for a very long time. While all L2O-based optimizers go past this optimum, it is noticeable that the guarding mechanism in the original GL2O algorithm slows down its convergence. Compared to that, LGL2O converges as fast as L2O while maintaining reasonable asymptotic stability. And more impressively, after 50k steps, LGL2O significantly outperforms both it's constituent parts (L2O and SGDnm) showing that a judicious switching between the two provides synergistic gains.

4.2. Out-of distribution dataset, in-distribution optimizee

This experiment evaluates performance of the L2O optimizer on optimizing an MLP on out-of-distribution, simple 2D datasets: Circles & Spirals.

On the Circles dataset, LGL2O behaves as expected and keeps converging in later stages of the rollout. Here, LGL2O is more noisy in the later stages than GL2O. One hypothesis is that this could be caused by noise in the optimizer fitness evaluation and could be fixed by averaging loss values over more mini-batches[2]. In fact, this suggest a future improvement we could make to the LGL2O algorithm 2, to adaptively increase n_t and n_c (keeping the ratio $n_t : n_c$ constant not to increase the number of function evaluations per step) when the mean plus or minus the standard deviation of $\left\{ \mathcal{F}(y^{k+n_t}, v_j) \right\}_{k \in [1, n_c]}$ and $\left\{ \mathcal{F}(z^{k+n_t}, v_j) \right\}_{k \in [1, n_c]}$ start to overlap in line 16 of algorithm 2.

The experiment on Spirals (see Figure 5) illustrates the incredible synergy operated by LGL2O. By judiciously alternating between its constituent parts (L2O and SGDnm), LGL2O is able to significantly outperform both does constituent parts after 50k optimization steps.

4.3. In-distribution dataset, out-of-distribution optimizee

This experiment illustrates performance of L2O and LGL2O on MNIST and a convolution neural network (CNN) as an optimizee. The optimizee is a 3-layer CNN with $number\_of\_channels = (8, 16, 32)$, $kernel\_sizes = (5, 3, 3)$ and $strides = (2, 2, 2)$, with a fully connected final layer.

Since the CNN optimizee is sufficiently different from the MLP that L2O was meta-trained on, L2O diverges very quickly, as seen in Figure 6. GL2O diverged as well[3], while LGL2O engages the guard soon enough and manages to converge steadily tracking it's fallback optimizer (SGDnm) as one would hope in the case where the learned optimizer is not helpful.

It is worth noting that while GL2O is guaranteed to converge asymptotically, practical performance in this case very bad. LGL2O however does the best one could expect in a bad situation.

---

[2]Both optimizers do $n_t = 10$ consecutive steps, then their loss is computed as a mean over $n_c = 10$ new mini-batches.

---

[3]This might be fixed by further tuning of the guarding algorithm hyperparameters but our grid search did not find good hyperparameters for GL2O in this case.
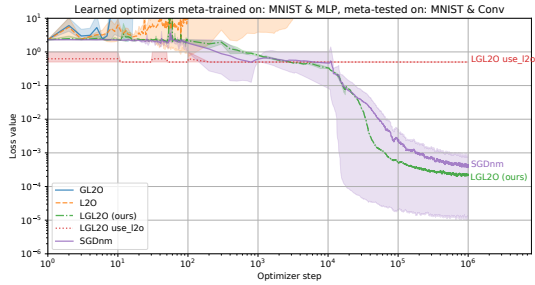
Figure 6. **In-distribution dataset, out-of-distribution optimizee.** L2O was meta-trained to optimize an MLP, therefore it does not generalize to optimization of CNNs here. LGL2O detects bad behaviour of the learned optimizer and switches to the fallback optimizer, which then performs similarly to the baseline, as one would hope.

4.4. Out-of-distribution dataset, out-of-distribution optimizee

This setup evaluates L2O and LGL2O on an out-of-distribution dataset (CIFAR-10) and an out-of-distribution optimizee (ConvNet). Results can be seen in Figure 7.
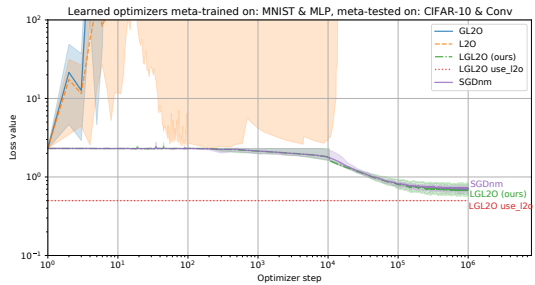

Figure 7. **Out-of-distribution dataset and out-of-distribution optimizee.** L2O diverges very quickly (due to being out of distribution for both the dataset and optimizee), while LGL2O converges using the fallback optimizer from the beginning.

In this case, again, L2O and GL2O completely fail (diverge very quickly to high loss values). While LGL2O uses its fallback optimizer from the beginning and saves the stable convergence.

4.5. Out-of-distribution dataset (Imagenet), out-of-distribution optimizee

The last type of setup evaluates L2O, GL2O and our LGL2O on an out-of-distribution dataset (TinyImagenet) and an out-of-distribution optimizee (ConvNet). For the results, a subset of the dataset, which contains randomly chosen 50 (out of full 200) classes was used. Considering

the nature of the dataset, a deeper convolutional network was used this time, namely: the optimizee is a 5-layer CNN with $number\_of\_channels = (8, 16, 32, 32, 64)$, $kernel\_sizes = (3, 3, 3, 3, 1)$ and $strides = (2, 2, 2, 1, 1)$, with a fully connected final layer. In order to achieve a stable convergence of the baseline (and the fallback optimizer at the same time), a grid-search for algorithm hyper-parameters was concluded (see the Fig.10 for reference). The resulting hyper-parameters used for the Tiny-Imagenet experiment are: $learning\_rate = 0.01$ and $learning\_rate\_decay = 50000$, these are used for SGDnm baseline and fallback optimizers for LGL2O and GL2O algorithms.

Results can be seen in Figure 8, the L2O diverges towards very high loss values, the GL2O diverges immediately to Not-a-Number loss values (therefore not shown in the graph), while our LGL2O correctly detects the tendency to L2O to diverge and switches to the fallback optimizer. As a results, it behaves almost identically to the SGDnm baseline, as desired.
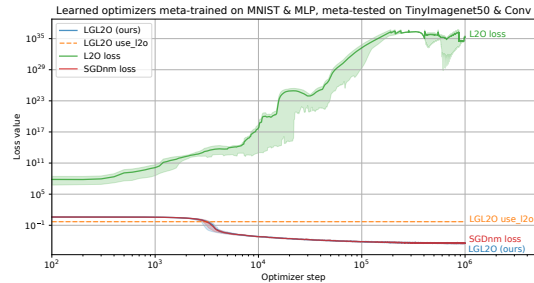

Figure 8. **Out-of-distribution dataset and out-of-distribution optimizee.** Here, the optimizee (network) has different nature (MLP vs ConvNet) but it has also different size than in the previous experiment. The GL2O algorithm diverges quickly to NaN loss values (ommitted in the graph), L2O diverges to high loss values, while ours, LGL2O, discovers the divergent tendency of the learned L2O and switches immediately to the fallback optimizer. Therefore the L2L2O behaves almost identically to the SGDnm baseline.

4.6. Faster convergence - Adam as the fallback optimizer of LGL2O

In many results shown above, the Adam baseline performed better than LGL2O in the long run (though for sample-efficiency in the beginning LGL2O outperformed Adam in cases where L2O generalized). This is caused by the fact that LGL2O was using SGD (without momentum) as a fallback optimizer. This experiment shows that by using the Adam optimizer as a fallback optimizer, it is possible to achieve considerably better results than with SGD (without momentum), see Figure 9. The graph shows convergence
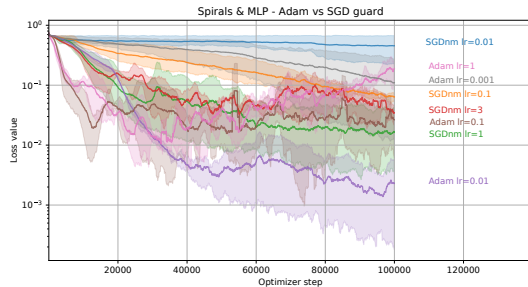
Figure 9. **LGL2O with a variety of different fallback optimizer.** In this experiment with an out-of-distribution dataset and in-distribution optimizee, we see that we can improve the practical performance of LGL2O by using Adam as a fallback optimizer instead of SGD. The legend designate which fallback optimizer was used in LGL2O as well as its learning rate (lr). **SGDnm: SGD without moment**. **SGD: SGD with moment**. **Adam: Adam**

of LGL2O optimizing an MLP optimizee on the Spirals dataset.

This paper focused on using an SGD fallback optimizer, the fallback optimizer needs to itself have a convergence guarantee in order to transmit that property to the LGL2O.

5. Conclusion

Most types of learned optimizers have common problems, they either fail to generalize outside of the training distribution and/or do not have good asymptotic properties (e.g. they are unable to overfit to a small dataset or get stuck on a suboptimal solution).

It has been shown that these drawbacks can be addressed using an explicit mechanism (i.e. guarding) that combines the good initial behaviour of learned algorithms with the desired asymptotic properties of analytic algorithms (like SGD).

This work builds on top of GL2O, which hybridizes learned and hand-crafted learning algorithms and combines best of both worlds. The LGL2O algorithm proposed in this paper is conceptually simpler (e.g. the guarding mechanism has fewer hyperparameters), and computationally less expensive, while maintaining convergence guarantees. We also show that it performs better in practice.

It was shown that LGL2O behaves as ideally desired, it converges quickly using the learned L2O initially and then converges steadily in later stages by relying on the hand-crafted guarding algorithm. At all times and in all cases, LGL2O performed as well or better than it's constituent parts (the learned optimizer, and the fallback optimizer SGD without momentum). And at times it was considerably better

than both. The same cannot be said of GL2O which while also provably convergent asymptotically, suffered significant performance loss compared to L2O at the beginning of training in the cases where the L2O was in-distribution or generalized.

In addition to having a simpler decision rule than the GL2O algorithm, LGL2O seems to work more robustly, especially in the ConvNet experiments.

Due to its simplicity, the algorithm did not need much explicit hyperparameter tuning.

The contributions of the paper are the following: We propose a new type of guard, LGL2O which is conceptually simpler than the exist class of guards (GL2O), has smaller time complexity, has fewer hyperparameters, and converges better in practice. We prove convergence guarantees for LGL2O. And finally we show that in practice it performs better than GL2O and vanilla L2O. In particular, we show LGL2O can allow the use of learned optimizers without divergence for millions of optimization steps which up until now could never exceed thousands of steps.

In general, the topic of hybridization of the learned algorithm was shown on the case of simple L2O, but the mechanisms shown here are completely general and it should be beneficial to apply them to other, more complex, meta-learning architectures. Further investigation in these directions is left for future work.

Acknowledgements

References

Alet*, F., Schneider*, M. F., Lozano-Perez, T., and Kaelbling, L. P. Meta-learning curiosity algorithms. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BygdyxHFDS.

Andrychowicz, M., Denil, M., Colmenarejo, S. G., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and de Freitas, N. Learning to Learn by Gradient Descent by Gradient Descent. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pp. 3988–3996, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. event-place: Barcelona, Spain.

Chen, T., Chen, X., Chen, W., Heaton, H., Liu, J., Wang, Z., and Yin, W. Learning to Optimize: A Primer and A Benchmark. mar 2021. URL http://arxiv.org/abs/2103.12828.

Co-Reyes, J. D., Miao, Y., Peng, D., Real, E., Le, Q. V., Levine, S., Lee, H., and Faust, A. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=0XXpJ4OtjW.

Gower, R. M. Convergence theorems for gradient descent, 2019. URL https://gowerrobert.github.io/pdf/M2_statistique_optimisation/grad_conv.pdf.

Heaton, H., Chen, X., Wang, Z., and Yin, W. Safeguarded Learned Convex Optimization. *arXiv:2003.01880*, 2020. URL http://arxiv.org/abs/2003.01880.

Hochreiter, S. and Jurgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to Learn Using Gradient Descent. In *Artificial Neural Networks — ICANN 2001*, pp. 87–94, 2001.

Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 2021. ISSN 1939-3539. doi: 10.1109/TPAMI.2021.3079209.

Kirsch, L. and Schmidhuber, J. Meta learning backpropagation and improving it. *Meta Learning Workshop at Advances in Neural Information Processing Systems*, 2020.

Kirsch, L., van Steenkiste, S., and Schmidhuber, J. Improving generalization in meta reinforcement learning using learned objectives. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1evHerYPr.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, 2009.

LeCun, Y. and Cortes, C. MNIST handwritten digit database. 2010. URL http://yann.lecun.com/exdb/mnist/.

Lv, K., Jiang, S., and Li, J. Learning Gradient Descent: Better Generalization and Longer Horizons. mar 2017. URL http://arxiv.org/abs/1703.03633.

Maheswaranathan, N., Sussillo, D., Metz, L., Sun, R., and Sohl-Dickstein, J. Reverse engineering learned optimizers reveals known and novel mechanisms. *arXiv:2011.02159*, November 2020. URL http://arxiv.org/abs/2011.02159.

Metz, L., Maheswaranathan, N., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. 2020. URL http://arxiv.org/abs/2009.11243.

Metz, L., Maheswaranathan, N., Freeman, C. D., Poole, B., and Sohl-Dickstein, J. Overcoming barriers to the training of effective learned optimizers. 2021. URL https://openreview.net/forum?id=MCe-j2-mVnA.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Prazeres, M. O. Stochastic gradient descent: An intuitive proof, 2020. URL https://medium.com/oberman-lab/proof-for-stochastic-gradient-descent-335bdc8693d0.

Prokhorov, D., Feldkarnp, L., and Tyukin, I. Adaptive behavior with fixed weights in RNN: an overview. In *Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02*, volume 3, pp. 2018–2022, 2002.

Rosa, M., Afanasjeva, O., Andersson, S., Davidson, J., Guttenberg, N., Hlubuček, P., Poliak, M., Vítků, J., and Feyereisl, J. BADGER: Learning to (Learn [Learning Algorithms] through Multi-Agent Communication). 2019. URL https://arxiv.org/abs/1912.01513.

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.

Sandler, M., Vladymyrov, M., Zhmoginov, A., Miller, N., Madams, T., Jackson, A., and Arcas, B. A. Y. Meta-learning bidirectional update rules. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 9288–9300, 2021.

Sutton, R. The bitter lesson, 2019. URL http://incompleteideas.net/IncIdeas/BitterLesson.html.

Tan, M. and Le, Q. Efficientnetv2: Smaller models and faster training. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 10096–10106. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/tan21a.html.

Wu, Y., Ren, M., Liao, R., and Grosse, R. Understanding Short-Horizon Bias in Stochastic Meta-Optimization. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=H1MczcgR-.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. aug 2017. URL http://arxiv.org/abs/1708.07747.

Xu, Z., Dai, A. M., Kemp, J., and Metz, L. Learning an adaptive learning rate schedule. *CoRR*, abs/1909.09712, 2019. URL http://arxiv.org/abs/1909.09712.

Zhou, X. On the fenchel duality between strong convexity and lipschitz continuous gradient. *arXiv*, 2018. URL http://arxiv.org/abs/1803.06573. arXiv: 1803.06573.

A. Appendix

This appendix contains the proofs of convergence guarantees, the hyperparameters used and how they were obtained, how randomness was used in the algorithms, and which hardware and software were used as well as addition experiments of the LGL2O Guard with a different learned optimizer to show the generality of our guard.

Table of Contents:
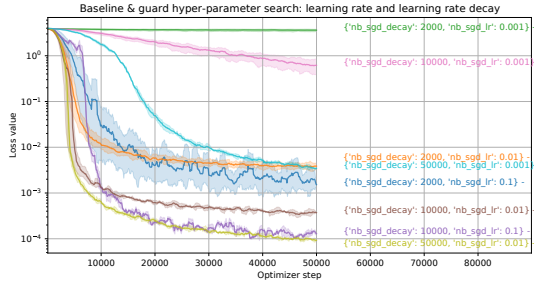
B. Additional Figures

Figure 10. Since the ConvNet optimizee has different topology and size than other methods, additional grid-search for the baseline and the guard hyper-parameters (SGDnm) was made. Results are concluded over 5 runs. The resulting hyperparameters used for the TinyImagenet experiment are: $learning\_rate = 0.01$ and $learning\_rate\_decay = 50000$.

C. Proof of Convergence Guarantee

Many of the proofs in this section have been strongly inspired by (Prazeres, 2020) and (Gower, 2019)

Proposition 2. *Let f be a continuous loss function which is L-smooth and let w^* be its global minimum. Let $E(w) := f(w) - f(w^*)$. Let w_{i+1} and w_i be two subsequent points in a sequence generated by the Loss-Guarded L2O algorithm using (deterministic) gradient descent with a step size of α/L for the guarding mechanism. Then:*

$$E(w_{i+1}) - E(w_i) \leq -\frac{\alpha}{L}(1 - \frac{\alpha}{2})\|\nabla f(w_i)\|_2^2 \quad (2)$$

and

$$E(w_i) \geq \frac{\alpha}{L}(1 - \frac{\alpha}{2})\|\nabla f(w_i)\|_2^2 \quad (3)$$

Proof. If w_{i+1} is chosen via the guarding mechanism then we have that $w_{i+1} = w_i - \frac{\alpha}{L}\nabla f(w_i)$. Furthermore by virtue of f being L-smooth we have that

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|_2^2. \quad (4)$$

Thus assuming that w_{i+1} is chosen via the guarding mechanism we have that

$$\begin{aligned}
E(w_{i+1}) - E(w_i) &= (f(w_{i+1}) - f(w^*)) - (f(w_i) - f(w^*)) \\
&= f(w_{i+1}) - f(w_i) \\
&=^\dagger f\left(w_i - \frac{\alpha}{L}\nabla f(w_i)\right) - f(w_i) \\
&\leq^\ddagger \left[f(w_i) - \frac{\alpha}{L}\langle \nabla f(w_i), \nabla f(w_i) \rangle + \right. \\
&\quad \left. + \frac{L}{2}\frac{\alpha^2}{L^2}\|\nabla f(w_i)\|_2^2\right] - f(w_i) \\
&= -\frac{\alpha}{L}(1 - \frac{\alpha}{2})\|\nabla f(w_i)\|_2^2.
\end{aligned}$$

Where for $\dagger$ we use the fact w_{i+1} is obtained through the guarding mechanism, and for $\ddagger$ we applied inequality (4) with $y = w_i - \frac{\alpha}{L}\nabla f(w_i)$ and $x = w_i$. This proves inequality (2) if w_{i+1} was chosen using the guarding.

If w_{i+1} was chosen using L2O, then we know by criterion (1) that

$$f(w_{i+1}) < f\left(w_i - \frac{\alpha}{L}\nabla f(w_i)\right)$$

and so the above derivation follows through except that at $\dagger$ we have a (strict) inequality instead of an equality.

Inequality (3) is easily obtained from (2) by noticing that $\forall w, E(w) \geq 0$ because w^* is the global minimum of f, and so $-E(w_i) \leq E(w_{i+1}) - E(w_i)$. $\square$

Theorem 3. *Let f be a continuous loss function which is strictly convex and L-smooth and let w^* be it's global minimum. Then $E(w) := f(w) - f(w^*)$ is a Lyapunov function for sequences of points generated by the Loss-Guarded L2O algorithm using (deterministic) gradient descent for the guarding mechanism with a step size of $\frac{a}{L}$ with $\alpha \in]0, 2[$.*

Proof. To show that E is a Lyapunov function we need to show that:

1. E is continuous.

2. $E(w) = 0$ if and only if $w = w^*$.

3. $E(w) > 0$ if and only if $w \neq w^*$.

4. $E(w_{i+1}) \leq E(w_i)$, $\forall i \in \mathbb{N}$.

1. is automatic by continuity of f. 2 and 3 and also immediate by virtue of w^* being the unique global minimum of f. And finally 4 is give from inequality (2) of Proposition 2. □

Lemma 4. *Strong convexity implies the Polyak-Łojasiewicz Condition (Zhou, 2018):*

$$E(w) := f(w) - f(w^*) \leq \frac{1}{2\mu}\|\nabla f(w)\|_2^2. \quad (5)$$

Proof. From strong convexity we have that

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|_2^2.$$

Using Hölder's inequality we have that

$$f(y) \geq f(x) - \|\nabla f(x)\|_2 \|y - x\|_2 + \frac{\mu}{2}\|y - x\|_2^2.$$

Let y_L be the value of y which minimizes the left hand side of the inequality and y_R be the value that minimizes the right hand side of the inequality. Then we have

$$\begin{aligned}
f(y_L) &\geq f(x) - \|\nabla f(x)\|_2^2 \|y_L - x\|_2^2 + \frac{\mu}{2}\|y_L - x\|_2^2 \\
&\geq f(x) - \|\nabla f(x)\|_2 \|y_R - x\|_2 + \frac{\mu}{2}\|y_R - x\|_2^2.
\end{aligned} \quad (6)$$

Minimizing (6) with respect to y_R we find that y_R satisfies:

$$\|y_R - x\|_2 = \frac{1}{\mu}\|\nabla f(x)\|_2. \quad (7)$$

Now using the fact that by definition $y_L = x^*$, the minimum of f, and plugging (7) in (6) we get

$$f(x^*) \geq f(x) - \frac{1}{2\mu}\|\nabla f(x)\|_2^2,$$

which can be re-arranged to give the desired Polyak-Łojasiewicz Condition. □

> **Theorem 5.** *Let f be a continuous loss function which is μ-strongly convex and L-smooth and let w^* be it's global minimum. Then sequences of points generated by the Loss-Guarded L2O algorithm using (deterministic) gradient descent for the guarding mechanism with a step size of $\frac{a}{L}$ with $\alpha \in]0, \min(2, 2\mu L)[$ converge to w^*, i.e.*
>
> $$\lim_{i \to \infty} w_i = w^*$$
>
> .

Proof. Let E be the Lyapunov function of Theorem 3. By μ-strong convexity of f, we have that f satisfies the Polyak-Łojasiewicz Condition (5). Thus from Proposition 2, we have that:

$$\begin{aligned}
E(w_{i+1}) - E(w_i) &\leq -\frac{\alpha}{L}(1 - \frac{\alpha}{2})\|\nabla f(w_i)\|_2^2 \\
&\leq -\frac{\alpha}{2\mu L}(1 - \frac{\alpha}{2})E(w_i).
\end{aligned}$$

Where the second inequality follows from the Polyak-Łojasiewicz condition. Thus:

$$E(w_{i+1}) \leq \left[1 - \frac{\alpha}{2\mu L}(1 - \frac{\alpha}{2})\right]E(w_i). \quad (8)$$

And thus for $\alpha \in]0, \min(2, 2\mu L)[$, convergence to w^* is guaranteed because (8) implies that

$$E(w_i) \leq \left[1 - \frac{\alpha}{2\mu L}(1 - \frac{\alpha}{2})\right]^i E(w_0),$$

which means that $\lim_{i \to \infty} E(w_i) = 0$. □

Definition 1. Given a loss function for $f : \mathbb{R}^k \to \mathbb{R}$ composed of the sum of the loss for each sample $x \in X$:

$$f(w) := \sum_{x \in X} \varphi(w, x)\mu_X(x),$$

where $\varphi : \mathbb{R}^k \times X \to \mathbb{R}$ is the sample-dependent loss function and $\mu_X(x)$ is the probability density of sample x, we define the mini-batch stochastic gradient of mini-batch size M of f at w, $\nabla_{mb}f(w)$, to be the random variable $\nabla_{mb}f(w) : X^M \to \mathbb{R}^k$ with probability distribution

$$P\left(\nabla_{mb}f(w) = \frac{1}{\sum_{i=1}^M \mu_X(x_i)}\sum_{i=1}^M \nabla_w \varphi(w, x_i)\mu_X(x_i)\right) = \\ = \prod_{i=1}^M \mu_X(x_i).$$

Proposition 6. *Let f be a continuous loss function which is L-smooth and let w^* be it's global minimum. Let $E(w) := f(w) - f(w^*)$. Let w_{i+1} and w_i be two subsequent points in a sequence generated by the Loss-Guarded L2O algorithm using mini-batch stochastic gradient descent with a step size of α/L for the guarding mechanism. If, in expectation, the stochastic gradient $\nabla_{mb}f(w)$ is equal to the true gradient,*

$$\mathbb{E}(\nabla_{mb}f(w)) = \nabla f(w),$$

and the variance of the stochastic gradient is bounded,

$$\text{Var}(\|\nabla_{mb}f(w)\|_2^2) \leq \sigma^2,$$

then:

$$\begin{aligned}
E(w_{i+1}) - E(w_i) \leq &-\frac{\alpha}{L}(1 - \frac{\alpha}{2}) \\
&\left(\|\nabla f(w_i)\|_2^2 - \frac{\alpha\sigma^2}{2(1 - \alpha/2)}\right) \quad (9)
\end{aligned}$$

and

$$E(w_i) \geq \frac{\alpha}{L}(1 - \frac{\alpha}{2})\left(\|\nabla f(w_i)\|_2^2 - \frac{\alpha\sigma^2}{2(1-\alpha/2)}\right) \quad (10)$$

Proof. The proof follows as in Proposition 2, except

that we now do it in expectation (over the mini-batch) and that if the guarding mechanism is used, then $w_{i+1} = w_i - \frac{\alpha}{L}\nabla_{mb}f(w_i)$, but as per criterion (1), whether L2O is used or the guarding mechanism, we have that $f(w_{i+1}) \leq f(w_i - \frac{\alpha}{L}\nabla_{mb}f(w_i))$. Thus assuming that w_{i+1} is chosen via the guarding mechanism we have that

$$\begin{aligned}
\mathbb{E}\left[E(w_{i+1}) - E(w_i)|w_i\right] &= \left(\mathbb{E}\left[f(w_{i+1})|w_i\right] - f(w^*)\right) - \left(f(w_i) - f(w^*)\right) \\
&= \mathbb{E}\left[f(w_{i+1})|w_i\right] - f(w_i) \\
&\leq \mathbb{E}\left[f\left(w_i - \frac{\alpha}{L}\nabla_{mb}f(w_i)\right)\Big|w_i\right] - f(w_i) \\
&\leq^\dagger \left\{f(w_i) - \frac{\alpha}{L}\langle\nabla f(w_i), \mathbb{E}\left[\nabla_{mb}f(w_i)|w_i\right]\rangle + \frac{L}{2}\frac{\alpha^2}{L^2}\mathbb{E}\left[\|\nabla_{mb}f(w_i)\|_2^2|w_i\right]\right\} - f(w_i) \\
&\leq^\ddagger \left\{f(w_i) - \frac{\alpha}{L}\langle\nabla f(w_i), \nabla f(w_i)\rangle + \frac{L}{2}\frac{\alpha^2}{L^2}\left(\|\nabla f(w_i)\|_2^2 + \sigma^2\right)\right\} - f(w_i) \\
&= -\frac{\alpha}{L}(1 - \frac{\alpha}{2})\left(\|\nabla f(w_i)\|_2^2 - \frac{\alpha\sigma^2}{2(1-\alpha/2)}\right),
\end{aligned}$$

where for † we applied inequality (4) with $y = w_i - \frac{\alpha}{L}\nabla_{mb}f(w_i)$ and $x = w_i$, and for ‡ we use the fact that $\mathbb{E}\left[\nabla_{mb}f(w_{i+1})|w_i\right] = \nabla f(w)$ and that since the variance of $\nabla_{mb}f(w)$ is bounded by σ^2 we have that $\mathbb{E}\left[\|\nabla_{mb}f(w_i)\|_2^2|w_i\right] \leq \|\nabla f(w_i)\|_2^2 + \sigma^2$.

Again inequality (10) follows trivially from (9) by noticing that $\forall w, E(w) \geq 0$ because w^* be is the global minimum of f, and so $-E(w_i) \leq E(w_{i+1}) - E(w_i)$. □

Theorem 7. *Let f be a continuous loss function which is μ-strongly convex and L-smooth and let w^* be it's global minimum. If, in expectation, the stochastic gradient $\nabla_{mb}f(w)$ is equal to the true gradient,*

$$\mathbb{E}(\nabla_{mb}f(w)) = \nabla f(w),$$

and the variance of the stochastic gradient is bounded, then sequences of points generated by the Loss-Guarded L2O algorithm using stochastic gradient descent for the guarding mechanism with a step size of $\frac{a_i}{L}$ with $\alpha_i \propto \frac{1}{i_0+i}$ converge to w^*, i.e.

$$\lim_{i\to\infty} w_i = w^*.$$

Proof. From Proposition 6 and μ-strong convexity (which gives us the Polyak-Łojasiewicz Condition (5)) we have that

$$\mathbb{E}\left[E(w_{i+1}) - E(w_i)|w_i\right] \leq -\frac{\alpha}{2\mu L}(1 - \frac{\alpha}{2})E(w_i) + \frac{\alpha^2\sigma^2}{2L}. \quad (11)$$

Which we can rewrite as

$$\begin{aligned}
\mathbb{E}\left[E(w_{i+1})|w_i\right] &\leq \left(1 - \frac{\alpha}{2\mu L}(1 - \frac{\alpha}{2})\right)E(w_i) + \frac{\alpha^2\sigma^2}{2L} \\
&\leq \left(1 - \frac{\alpha}{4\mu L}\right)E(w_i) + \frac{\alpha^2\sigma^2}{2L}, \quad (12)
\end{aligned}$$

where the second inequality is valid if $0 < \alpha < 1$ because then $(1 - \frac{\alpha}{2}) > \frac{1}{2}$.

We will now prove by recursion, using inequality (12) that with the appropriate learning rate $\lambda_i = \frac{\alpha_i}{L}$ we have the follow inequality for all i:

$$\mathbb{E}\left[E(w_{i+1})\right] \leq \frac{C}{i + i_0 + 1}, \quad (13)$$

for positive constants $C = 32\mu^2 L\sigma^2$ and i_0. Which implies that $\lim_{i\to\infty}\mathbb{E}\left[E(w_{i+1})\right] \to 0$ and thus that w converges to w^* in expectation.

For $i = 0$:

$$\mathbb{E}\left[E(w_0)\right] \leq \frac{C}{i_0} = E(w_0)$$

is true if we set $i_0 = C/E(w_0)$.

Now supposing inequality (13) is true up to i, let us show it

for $i + 1$ by using $\alpha_i := \min(1; 2\mu L; \frac{8\mu L}{i + i_0})$:

$$
\begin{aligned}
\mathbb{E}\left[E(w_{i+1})\right] &\leq \left(1 - \frac{\alpha_i}{4\mu L}\right)\mathbb{E}[E(w_i)] + \frac{\alpha_i^2 \sigma^2}{2L} \\
&\leq \left(1 - \frac{\alpha_i}{4\mu L}\right)\frac{C}{i + i_0} + \frac{\alpha_i^2 \sigma^2}{2L} \\
&\leq \left(1 - \frac{2}{i + i_0}\right)\frac{C}{i + i_0} + \frac{(8\mu L)^2 \sigma^2}{2L(i + i_0)^2} \\
&= \left(1 - \frac{2}{i + i_0}\right)\frac{C}{i + i_0} + \frac{C}{(i + i_0)^2} \\
&= C\frac{i + i_0 - 1}{(i + i_0)^2} \\
&\leq \frac{C}{i + 1 + i_0},
\end{aligned}
$$

where the last inequality uses $\frac{k-1}{k^2} \leq \frac{1}{k+1}$ $\qquad\square$

D. Experiments with Different Learned Optimizer

This paper tries to address a common problem of learned optimizers: they are not easily predictable and their behaviour can depend on the meta-testing procedure (e.g. on what optimizee and dataset they were meta-trained on). For this reason we show the same experiments as in section 4, but with a different learned optimizer (L2O) that was meta-trained on a **ConvNet** optimizee and the **Cifar10** dataset instead of on an MLP with MNIST as in section 4. The purpose of these experiments is to:

- show that our method (LGL2O) works independently of which learned optimizer is used and that

- LGL2O acts appropriately in all situations and provides the best of both the learned optimizers and the general optimizer.

These experiments follows exactly the setup shown in the Experiments section 4. First, the L2O optimizer was meta-trained to optimize the ConvNet optimizee on the Cifar10 dataset. All the experiment parameters and hyperparameters follow the original experiments exactly, except of the learning rate of the meta-optimizer set to $lr = 0.0001$. No fine-tuning was made in order to obtain these results. The meta-training progress is shown in the Figure 11.

For clarity, the Table 2 shows the meta-testing experiment settings.

D.1. Discussion

Here, the same meta-training procedure was used on a different optimizee and dataset. At the beginning of the meta-training (see Fig. 11), the L2O optimizer weights
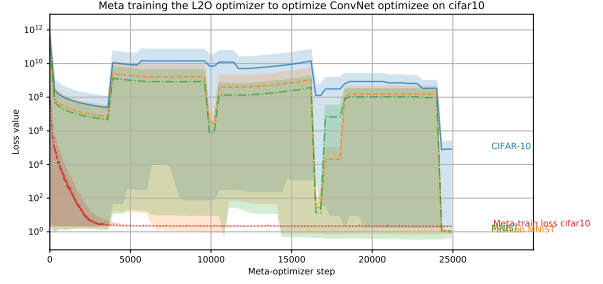

Figure 11. Outer-loop convergence of the learned optimizer on the Cifar10 dataset. The L2O method converges to sub-optimal, but stable results on the train set. During meta-testing, it is apparent that variance between test runs is high, but the optimizer performs comparably well on the test-set and generalizes the behaviour to other datasets. This meta-testing procedure shows only subset of datasets where ConvNet optimizee can be used. Mean and min-max ranges are across 5 runs are shown.

In Distribution?		Experiment Type	
Dataset	**Optimizee**	**Dataset**	**Optimizee**
no	no	MNIST	MLP
no	yes	MNIST	Conv
no	no	Spirals & Circles	MLP
yes	yes	CIFAR-10	Conv

Table 2. Overview of the experiments with updated in/out-distribution settings: after the meta-training phase, the ability of the hybrid optimizer LGL2O is evaluated on long rollouts, on in-distribution and out-of-distribution datasets and optimizees.

were initialized randomly. Probably due to the fact that the weights of ConvNet optimizee are shared, the implications of the weight modifications proposed by random L2O on the loss value are big and therefore the loss values explode in just first few optimizer steps. Gradually, the meta-training changes the optimizer to optimize the weights to a stable performance. Despite the fact that absolute values of the loss are rather far from optimal, it can be seen that the optimizer generalizes similar results to the in-distribution meta-test set and to other datasets.

Then, the learned L2O optimizer[4] parameters were used for meta-testing on long rollouts.

It can be seen that the learned optimizer generalizes from the ConvNet optimizee to **MLP optimizee** very well. It does not tend to diverge much in very long rollouts, despite that it was trained only on rollouts that were 600 steps long. In these cases, our method (LGL2O) performs better or at least

---

[4] One of the meta-training runs that achieved good results on Cifar10 at the end.
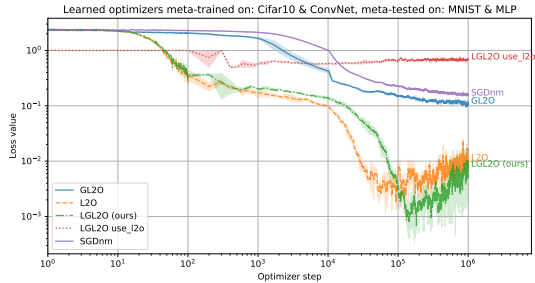
Figure 12. **Out-of-distribution dataset, out-of-distribution optimizee.** MLP optimizee optimized by different optimizers on the MNIST dataset. The L2O optimizer meta-trained on ConvNet produces very stable behaviour which starts to diverge after around 10^5 steps. Our LGL2O optimizer is slightly slower, achieves better performance and then starts slowly diverging as well. This is probably caused by noise in the switching mechanism, since it can be seen that the use_l2o indicates that the guard is switching at high frequency.

equally well compared to the L2O. It often outperforms the GL2O and it always outperforms the baseline SGNnm. This can be seen especially in the Fig. 14, where the SGDnm gets stuck in local optimum, L2O starts to plateau after 10^5 steps, but our LGL2O switches to the fallback optimizer at this point to continue reducing the loss. LGL2O switched at the right time (i.e. after the local optimum was avoided) and continues converging at a faster rate than L2O after that.

This is obvious in retrospect, but for LGL2O to work, the fallback optimizer cannot diverge. Initially we had used too large a learning-rate for the fallback optimizer which caused LGL2O to explode when it decided to switch the the fallback optimizer when the optimizee was a ConvNet [5].

Of particular note is the divergence of LGL2O in Fig. 13. This is exactly the risk of which we foretold in the last the paragraph of section 3 and which me mentioned again in the paragraph on the Circles dataset in section 4.2. Close to the minimum of the loss function, 10 mini-batches is not enough anymore to reliably distinguish between the losses given by the updates proposed by the learned optimizer and the fallback optimizer. We can see that that is the problem because of the high frequency switching between learned optimizer and fallback optimizer after 10^5 steps (green line in Fig. 13). As previously mentioned, at that point (past 10^5 updates in this case) one would need to increase the hyperparameter n_c. Doing so in adaptive manner as suggested in section 4.2 would probably be best. But in all experiments in this paper, we did not hyperparameter search for the Guard hyperparameters and simply always used the

---

[5] For this reason, graphs in Figures 13 and 16 use SGD $lr = 0.0001$ instead of $lr = 0.01$ and 100x slower learning rate decay.
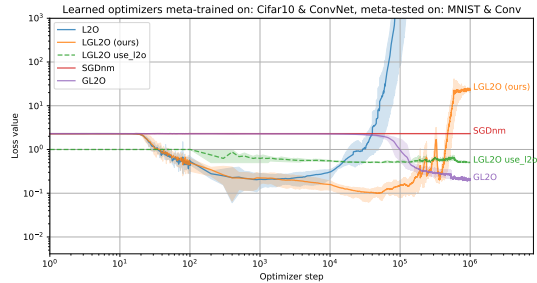

Figure 13. **Out-of-distribution dataset, in-of-distribution optimizee.** ConvNet optimizee optimized by different optimizers on the MNIST dataset. Here, the L2O diverges significantly, the GL2O counters the tendency of the learned optimizer to diverge. The LGL2O diverges significantly because criterion 1 is not reliably evaluated anymore as can be seen by high frequency switching between learned optimizer and fallback optimizer after 10^5 steps (green line). Close to the minimum of the loss function, 10 mini-batches is not enough anymore to reliably distinguish between the losses given by the updates proposed by the learned optimizer and the fallback optimizer. Until then however, LGL2O improved significantly on both it's constituent parts.

same ones $n_t = n_c = 10$.

E. Hyperparameters

Hyperparameters for baseline learning algorithms (as well as guards) were partially chosen from experience and partially found by grid search. In all experiments, the guard has hyperparameters identical to the corresponding baseline - this is SGDnm in most cases. In all cases, our batch-size was 128 samples per mini-batch.

Hyperparameters of Adam were found for each combination of optimizee (MLP or ConvNet) and dataset found from the set [0.0001, 0.001, 0.01, 0.1] over 300 optimization steps. In case of SGD, the learning rate was set based on practical experience to 3.0 and (the optional) momentum to 0.9.

In order to fulfill assumptions of convergence guarantee, the following decaying learning rate schedule was used:

$$\text{lr}(t) = \frac{\text{lr}_0}{\left(\frac{t}{\text{decay-time-scale}} + 1\right)^{1.5}},$$

where t is the optimization step number and where the power of 1.5 was chosen because for convergence guarantees for SGD the power must be strictly greater than 0 and strictly less than 2, and 1.5 is a number in that range.

For SGDnm, grid search for each combination of optimizee and dataset was performed to determine the value of *decay-time-scale* from the following set [2000, 5000,
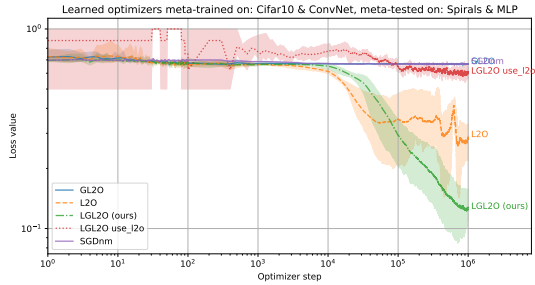
Figure 14. **Out-of-distribution dataset, out-of-distribution optimizee.** MLP optimizee optimized by different optimizers on the Spirals dataset. The Spirals dataset has a strong local optimum if used with a small MLP. In this case, the SGDnm and GL2O get stuck, while the L2O and LGL2O avoid the local optimum and converge well. In this case, our LGL2O outperforms L2O and keeps converging steadily.
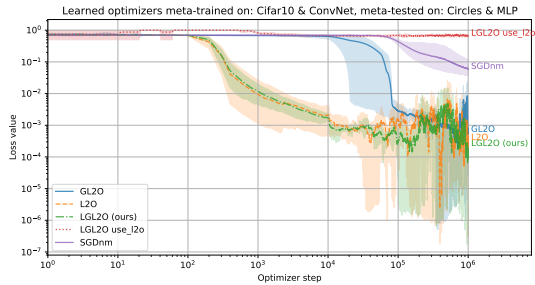

Figure 15. **Out-of-distribution dataset, out-of-distribution optimizee.** MLP optimizee optimized by different optimizers on the Circles dataset. In this case, all the learned optimizers are faster than the SGDnm baseline and they start to oscillate near optimal loss values. The L2O and LGL2O are faster than GL2O.

10000, 20000, 50000], over 300 optimization steps, the same value of *decay-time-scale* was used also for the SGD. LGL2O, the *decay-time-scale* was set a constant value of 50'000 for all experiments without doing a grid search; that value was arrived at simply because SGD tended to plateau roughly at the $50'000^{th}$ time step when running experiments at constant learning rates.

The best hyperparameters found using grid-search and then used as baselines (and optionally guards) were:

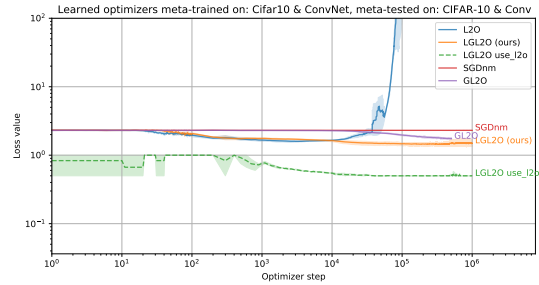|	Adam lr	SGDnm decay
MLP MNIST	0.001	50'000
Conv MNIST	0.01	20'000
MLP Circles	0.01	50'000
MLP Spirals	0.01	50'000
Conv CIFAR10	0.001	20'000

Figure 16. **In-distribution dataset, in-distribution optimizee.** ConvNet optimizee optimized by different optimizers on the Cifar10 dataset. Here, L2O optimizes well for many steps (as one would hope, being in distribution), but then it gets unstable. The GL2O deals with the instability and works as expected. The LGL2O works even better by benefiting from the excellent performance L2O and then slowly switching to it's fallback optimizer before L2O would start getting unstable. SGDnm, the fallback optimizer, in contrast has very poor performance as a standalone optimizer.

The gradients of the L2O LSTM were truncated every 20 steps. In LGL2O 10 steps of L2O were compared with 10 steps of SGD on 10 "validation" mini-batches (pulled from the training data but different from the mini-batches on which the 20 steps were just made) and whichever did better on those 10 validation mini-batches got to update the optimizee. This helped avoiding effect of noise in the fitness evaluation and increased the stability of the LGL2O algorithm significantly.

For GL2O we used $\alpha = 0.99$ $\theta = 0.9$, $m = 30$, and used the exponentially moving average series.

F. Randomness

Each experiment was run using 5 randomly chosen seeds. All neural network were initialized using the default PyTorch initializations.

G. Hardware and Software

All experiments were code in Python 3.9 with PyTorch 1.8.1 on CUDA 11.0. Every run was run on a single NVIDIA GPU with a memory of between 4Gb and 12 Gb. Running one seed of all the optimization algorithms on one dataset takes roughly 5 hours of wallclock time depending on the specific GPU and usage level of the machine it is running on. The SciKit learn datasets were loaded from SciKit-Learn version 0.24.0.