Wildauer Konferenz für Künstliche Intelligenz WiKKI - Wildau 2025
Beiträge zur Wildauer Konferenz für Künstliche Intelligenz 2025
https://doi.org/10.52825/th-wildau-ensp.v2i.2939

Published: 12 Sep. 2025

A Novel Exponential Continuous Learning Rate Adaption Gradient Descent Optimization Method

Alexander Kleinsorge[1,*], Alexander Fauck[1,2], and Stefan Kupper[1,2]

[1]Technische Hochschule Wildau, Germany

[2]TIB Open Publishing, Germany

*Correspondence: Alexander Kleinsorge, alexander.kleinsorge@th-wildau.de

Abstract. We present two novel, fast gradient based optimizer algorithms with dynamic learning rate. The main idea is to adapt the learning rate α by situational awareness, mainly striving for orthogonal neighboring gradients. The method has a high success and fast convergence rate and relies much less on hand-tuned hyper-parameters, providing greater universality. It scales linearly (of order $O(n)$) with dimension and is rotation invariant, thereby overcoming known limitations. The method is presented in two variants C2Min and P2Min, with slightly different control. Their impressive performance is demonstrated by experiments on several benchmark data-sets (ranging from MNIST to Tiny ImageNet) against the state-of-the-art optimizers Adam and Lion.

Keywords: Neural Network, Training, Optimizer

1. Introduction

Numerical optimization of functions obviously relies on data obtained from the function $f(x)$ landscape. One key problem is that we are lacking meaningful global information about $f(x)$ usually making it necessary to rely on local information instead. Approaches based on local properties range from using the function value in physics-inspired relaxation approaches [1], to algorithms using the topographical structure of the function landscape directly, such as gradient descent-like approaches, to biology inspired algorithms such as swarm optimization [2].

Among these, the gradient descent-like methods have the longest history and are (due to their linear scaling with the problems dimension) the only practically applicable algorithms in high dimensional problems (e.g. deep neural networks). In these approaches the gradient $G = \nabla f(x)$ of the function $f(x)$ is computed and thus also the best descent direction $-G$. However, while the idea of going downhill is obviously reasonable, an optimal step-length $\lambda = \alpha \cdot ||\nabla(f(x))||$ remains to be chosen cost-efficiently. The parameter α is the learning rate (or step size). Most current gradient based algorithms use a fixed learning rate α, which additionally may depend on time/steps t. This holds in particular for the Ada-family[1] of optimizers, widely used

[1]Such as: AdaGrad, RMSProp, AdaDelta, Adam, Lion, which all scale the gradient components individually (precondition-like).

for training neural networks. To eliminate the initial tuning of α, there are some modern approaches which adapt α dynamically, such as AdaDelta or Prodigy (proposed in [3]) or DoG (proposed in [4]). Yet, they perform not better then the most prominent Ada-optimizer Adam [5] or its successful predecessor Lion [6] and converge ultimately to constant α in most cases.

The use of a fixed learning rate α is in part due to the fact that it allows for precise mathematical analysis, guaranteeing or almost surely guaranteeing (for SGD) a lower bound on convergence rates (e.g. see [7], 1.2.3).

We propose a paradigm changing algorithm that estimates in each step ***self-consistently a near optimal*** learning rate α from low-cost local knowledge of the function, thereby achieving a jump close to the next minimum along the gradient direction. In particular, α approaches a problem-specific good scale exponentially fast and, depending on the problem, the adaption leads to continual changes of α.

We propose **ELRA** – **E**xponential **L**earning **R**ate **A**daption as a name for the class of optimizers, based on this idea.

Recent articles indicate that large variations of α might be very beneficial. In [8], it is for the first time mathematically proven that (periodically) varying step sizes lead to much better convergence rates, which our experimental results confirm. In [9] it is shown that estimating the best α via backtracking using Armijo's condition (see [7], 1.2.3) can lead to faster convergence then the Ada-family. However, each backtracking step needs a separate and expensive function value. Hence, backtracking more than once is seldom justified by the speed gained. ELRA does not suffer from this computational conundrum, as we provide two low-cost estimators for the best α, thereby retaining the benefit of a good α without losing speed.

The first essential advantage of ELRA is that a strongly adaptive α completely eliminates the need to find 'by hand' a good constant α for each specific problem. Secondly, most modern training schemes rely on decreasing α over time to achieve better test accuracy. Yet the best timing is a priori unknown and often determined by educated guesses. The strong performance of the ELRA optimizers C2Min and P2Min (cf. Fig. 7-12) shows that a strongly adaptive α needs no external timing. Thirdly, ELRA is invariant under orthogonal transformations of x, such as rotations, unlike the Ada-family, see [10]. Such an invariance is preferable not only for geometric optimization, [10], but also important near saddle points (see Results 3).

In addition to the learning rate, we propose also dynamic adaption of the momentum and the batch size and provide a kind of soft restart (necessary, as big α can lead to temporary instability). Moreover, we present a technique that can improve the final result, which we call boosting.

We are convinced that each of these features on its own can, to a varying degree, be also beneficial for other types of optimizers, like the Ada-family (see App. table 3 for a summary of their properties). This article is a further development of our original idea from preprint[11].

2. Dynamical Adaptions

This section explains how we dynamically control the learning rate α, the momentum, the batch size and soft restarts and explains boosting.

The code of the ELRA optimizers is online available via [12], using PyTorch.

2.1 Optimal α Updates via Orthogonal Gradients

All gradient descent methods for minimizing a function $f(x)$ boil down to the following update scheme for x

$$x_{t+1} = x_t - \alpha \cdot \Big((1-\beta) G_t + \beta M_t \Big), \tag{1}$$

where $G_t = \nabla f(x_t)$ is the gradient at x_t, M_t the momentum, β the ratio between G_t and M_t (possibly zero). For the Ada-family, α is essentially constant while G_t is not actually the gradient, but a component-wise modification, which is dynamically adapted. However, this leads to a dependency on the coordinate system and the speed of the algorithm depends heavily on the concrete representation of the data (see Fig. 3). Moreover α has to be chosen with care, either using past results or initial try and error runs.

We provide a completely new approach which overcomes many of these problems. The main idea is to use the cosine $\cos_t := \cos(\measuredangle(G_t, G_{t-1}))$ of the angle between the current gradient G_t and the previous gradient G_{t-1} to determine the adaptation of the learning rate α. A short proof of why and how this is reasonable can be given as follows: To find α, such that $x_t = x_{t-1} - \alpha G_t$ is a local minimizer to the differentiable[2] function f near a non-critical point x_{t-1} in the negative gradient direction $-G_{t-1}$, one considers $h(\alpha) := f(x_{t-1} - \alpha \cdot G_{t-1}) = f(x_t)$, which is $f(x_t)$ at the next point x_t, depending on the learning rate α. Differentiating h with respect to α yields:

$$\begin{aligned} h'(\alpha) &= \big\langle \nabla f(x_t), -G_{t-1} \big\rangle = -\big\langle G_t, G_{t-1} \big\rangle \\ &= -\cos\big(\measuredangle(G_t, G_{t-1})\big) \cdot ||G_t|| \cdot ||G_{t-1}||, \end{aligned} \tag{2}$$

where $\langle a, b \rangle$ denotes the scalar product and $||a|| = \sqrt{\langle a, a \rangle}$ the euclidean norm. Note that $h'(0) = -||G_{t-1}||^2$ is negative. This means that h, and hence f, decreases for small α. In fact, h decreases until it reaches a critical point $\alpha_{min} > 0$, where we have $h'(\alpha_{min}) = 0 \Leftrightarrow \cos_t = 0$. If α_{min} is a non-degenerate critical point of h, then h has necessarily a local minimum at α_{min} and thus also f in the direction of $-G_{t-1}$. This gives the following conclusion: For the optimal learning rate α, which leads locally to the smallest $f(x_t)$, the current and previous gradients G_t and G_{t-1} are orthogonal to each other or equivalently $\cos_t = 0$. Moreover if $\cos_t > 0$ then an increased α gives better results, while for $\cos_t < 0$ a smaller α is better. Figuratively speaking (cf. Fig. 1): If we see Zig-zag or anti-parallel steps we should decelerate, while for primarily parallel steps we should accelerate.

As α_{min} depends continuously on x_{t-1}, we can expect that the optimal α_t for x_t does not vary too much from the optimal α_{t-1} for x_{t-1}. This justifies the use of $\cos_t$ as an oracle for the next α_t. Note that $\cos_t$ is computational much cheaper than Armijo's condition, as no extra gradient/function values are needed.

There are infinitely many ways to use this result to update α, which can all be written in the form $\alpha_t = \alpha_{t-1} \cdot (1 + \cos_t \cdot g)$, where g can be any positive function. We use the following two competing update formulas for α, which are implemented in two distinct

[2] See Math. suppl. (5), why even for non-differentiable activation functions (e.g. ReLU), one can assume that f is smooth.

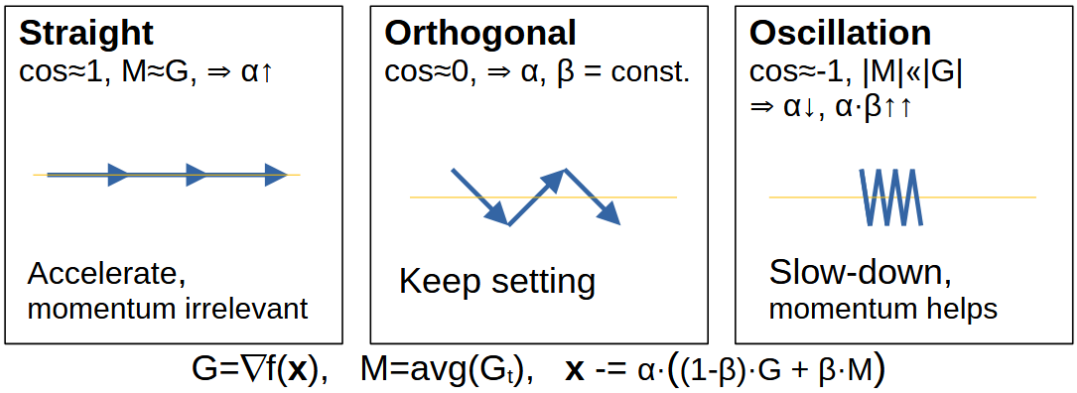


Figure 1. *Situations during optimization and associated α updates.*

optimizers **C2Min** and **P2Min**[3]:

$$\textbf{C2Min :} \qquad \alpha_t = \alpha_{t-1} \cdot \Big(1 + \max(0.5 \cdot \cos_t, 0.6 \cdot \cos_t)\Big)$$

$$\textbf{P2Min :} \qquad \alpha_t = \alpha_{t-1} \cdot \left(1 + \frac{\cos_t}{||G_{t-1}|| \,/\, ||G_t|| - \cos_t}\right) \cdot \kappa$$

The formula for C2Min leads to a mild exponential changes, as $0.5 \cdot \alpha_{t-1} \leq \alpha_t \leq 1.6 \cdot \alpha_{t-1}$, and has an asymmetric update behaviour, i.e. it decreases faster and increases slower, providing a more conservative and hence stable behaviour. The different weights 0.5 and 0.6 for negative/positive $\cos_t$ also improve stability of α in the case where $\cos_t$ oscillates lightly around 0, as $(1 - 0.5c)(1 + 0.6c)$ is slightly above 1 for $0 \leq c \leq 1/3$.

In the formula for P2Min, α_t is chosen such that $x_{t+1} = x_t - \alpha_t G_t$ is the minimizer of f along the line through x_{t-1} and x_t, if f were a parabola[4] in this direction (see Methods, A.2, for details). Note that the updated step size α_t can in principle be arbitrary between $-\infty$ and $+\infty$ for P2Min. We prevent this potentially catastrophic behaviour by imposing bounds of the form $0 < \alpha_t/\alpha_{t-1} < \gamma_{max}$, where γ_{max}[5] can be chosen at will, e.g. $\gamma_{max} \sim 10^6$. Moreover, we found that it is beneficial to set fixed bounds for α. Currently these bounds are set as follows: $10^{-8} < \alpha < 10^6$. These are additional hyper-parameters, yet they are sufficient in all our experiments.

An initial α_0 still has to be chosen for C2Min and P2Min. However, the specific choice is marginal, as both algorithms adapt α exponentially fast. We chose α_0 small (e.g. $\alpha_0 = 10^{-5}$) to prevent initial instabilities (explosions of $f(x)$). This leads to a negligible fixed number of initial extra steps to increase α to the right magnitude (see Fig. 4).

We remark that C2Min and P2Min are by construction rotation invariant[6], as they use only scalar products, and their computation is relatively cheap (effort of $O(n)$ for time and space), as computing scalar products (or norms and cosines) has a low computational resource demand.

[3]The factor $\kappa \sim 1$ neutralizes noise-effects in neural networks (see Methods, A.3, for details).
[4]The parabola ansatz is chosen, as near a local minima, each function is almost a parabola (cf. Math. Suppl. C).
[5]For neural networks, $\gamma_{max} = 10$ is probably sufficient, we use $\gamma_{max} = 10^6/(1+d)$, where $0 < d < 10^6$ is a damper, increased whenever we have a soft restart and decreased otherwise.
[6]Actually even invariant under orthogonal transformations.

2.2 Soft Restarts

The ELRA update-schemes for α can lead to numerical instabilities, due to overestimated increases for α (happening more often for P2Min, as it is more aggressive). To prevent a fatal increase of $f(x)$, we use retracing/soft restarts, if the new value $f(x_t)$ increases too much[7]. In these situations, we retrace back to the previous x_{t-2}, decrease α_t (at least $\alpha_t \leq 0.5{\cdot}\alpha_{t-1}$) and calculate the next point x_t by:

$$\begin{aligned} x_{t+1} &= x_t + (\alpha_{t-1} - \alpha_t)G_{t-1} \\ &= (x_t + \alpha_{t-1}G_{t-1}) - \alpha_t G_{t-1} = x_{t-1} - \alpha_t G_{t-1}. \end{aligned}$$

The question when to retrace is quite delicate and leaves room for much improvement. At the moment, we retrace if $f(x_t) > \min\big(25{\cdot}f(x_{best}), 1.1{\cdot}f(x_0)\big)$, where x_0 is the starting point and $f(x_{best})$ a moving average over the best function values. For the more mathematical experiments, such as Rosenbrock, we actually use for P2Min a more involved condition of the form $f(x_t) < f(x_{best})/D$, where D is a damping parameter, which increases whenever $f(x_t) > f(x_{t-1})$ and decreases otherwise.

2.3 Momentum Control

The optimizer P2Min uses **no** momentum, i.e. $M_t = 0$ or equivalently $x_{t+1} = x_t - \alpha_t G_t$ for P2Min**!**

C2Min on the other hand uses a simple decaying average of gradients as momentum $M_t = \delta{\cdot}M_{t-1} + (1{-}\delta){\cdot}G_{t-1}$, where we use $\delta = 0.8$. It is important to observe, that we use a different, dynamically controlled update parameter β_t for the actual step $x_t = x_{t-1} - \alpha_t\big((1{-}\beta_t)G_{t-1} + \beta_t M_{t-1}\big)$. By experiments, we found it beneficial to have $\beta_t < \delta$ and we control it by $\beta_t = 0.85 - 1.2\,\overline{o}_t$, where $\overline{o}_t = avg\big(\big|\frac{f(x)-\overline{f}}{\overline{f}}\big|\big)$ is the average relative oscillation of $f(x)$ around the mean function value $\overline{f}$ (see Methods, A.5, for details). We bound $\overline{o}$ by $0.043 \leq \overline{o} \leq 0.7$, to guarantee that $0.01 \leq \beta_t \leq 0.7984 < 0.8$.

2.4 Dynamical Batch Size

Influenced by the article "Don't Decay the Learning Rate, Increase the Batch Size" [13], we lately decided to also dynamically adapt the batch size b. We use integer multiples $m_t \geq 2$ of a minimal base batch size[8] b_{min}, i.e. $b_t = m_t \cdot b_{min}$. We always start with an initial batch size of $b_0 = 8{\cdot}b_{min}$, as it turned out to be beneficial for the whole optimization to first use a lager batch size, to let the optimizer find the right magnitude for α_t and a good optimization direction.

We update m_t every 100 steps, depending on the deviation of the gradients G_k within the t^{th}-batch from their mean G_t. To measure this deviation, we compute two gradients $G_1 = \sum_{k=1}^{b_t/2} G_k$ and $G_2 = \sum_{k=b_t/2+1}^{b_t} G_k$, representing each the gradient obtained from half of the batch. Then $s_t := \cos\big(\measuredangle(G_1, G_2)\big)$ estimates the cosine of the deviation angle between single gradients G_k within a batch and the batch gradient G_t. Taking the mean of s_t over 100 steps yields $\overline{s}$. We control the multiplier m_t (and thus b_t) via $\overline{s}$ by $m_t = \max\big(2, \lfloor m_{t-1}{\cdot}k_{t-1}\rfloor\big)$, where $k_{t-1} > 0$ for $\overline{s} \leq 0.015$, $k_{t-1} = 1$ for $0.015 < \overline{s} \leq 0.025$ and $k_{t-1} < 1$ for $\overline{s} > 0.025$ (see Methods, A.6, for details).

[7] One cannot avoid some increment of $f(x)$ with stochastic gradients, as G_t might not be the direction of steepest descent.

[8] b_{min} is a hyperparameter to be chosen for each problem.

We want to point out that s_t is calculated at a single point x_t and therefore independent of the optimizer, suggesting that this control parameter should also work for other optimizers, such as the Ada-family, an idea we deem worthy of investigation.

The constants 0.015 and 0.025 were obtained by experiments, however increasing the batch size b_t if $\overline{s} < \epsilon$ is mathematically justified by the fact that for $\overline{s} < \epsilon$ the stochastic gradients G_t are so noisy that on average they point in directions almost perpendicular to the actual gradient of $f(x_t)$, leading to random walks rather than gradient descent.

Finally, we want to draw the attention to the last batch of an epoch, which may contain fewer elements than the other batches. If this is the case, then we skip the last bacth, i.e. do not use it in this epoch, as it would yield a gradient that is much noisier than all others, thus often disrupting the optimization for some steps. This is in particular an issue for all optimizers which use larger learning rates α such as C2Min and P2Min.

2.5 Final Boosting

We noticed that C2Min and P2Min tend to oscillate round the optimal descent path (see Fig. 2). It follows that the mean $\overline{x} = \frac{1}{n}(\sum_{k=1}^{n} x_{T+k})$ of points x_t for a certain number n of steps (e.g. an epoch) can give better results, i.e. $f(\overline{x}) < f(x_{T+n})$. However, it is not beneficial within the optimization process to replace x_{T+n} with $\overline{x}$, as $\overline{x}$ is relatively to the optimal descent path still further up than x_{T+n} (in Fig. 2, $\overline{x}$ is roughly in the middle, while x_{T+n} is at the back, at the end of the green arrow). Yet in the final epochs, calculating $\overline{x}$ and $f(\overline{x})$ can boost the final result. We plot in our experiments below $f(\overline{x})$ for every epoch to illustrate the possible benefit.

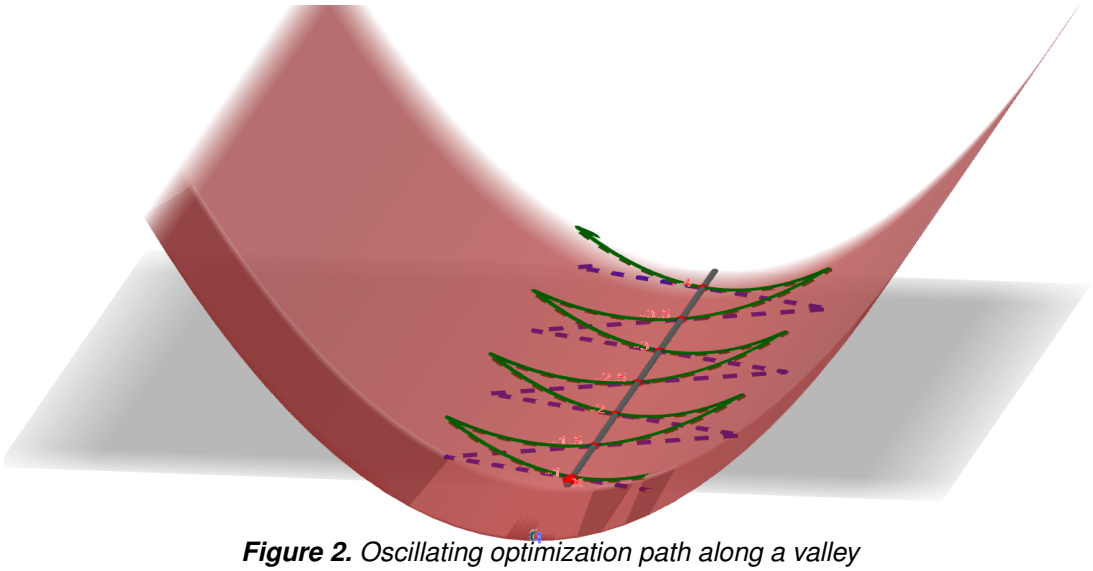


Figure 2. *Oscillating optimization path along a valley*

3. Results

As shown above (see section 2.1), we have a mathematical justification for our approach. Yet, giving guaranteed convergence rates for our proposed optimizers is intractable using current methods (even for convex landscapes), due to the adaptive nature of the learning rate α. Thus we rely on experiments to show the usefulness of ELRA. We conducted low dimensional mathematical experiments and high-dimensional experiments with neural networks for image classification. The latter are almost all executed for multiple random initializations, as gradient descent methods show partially chaotic behaviour. However, for cost reasons (limitations of an academical budget) we restrict ourselves

to 10 different initializations per experiment (except for Wide-ResNet on CIFAR-10 and Tiny ImageNet, where we conducted only 3 or single runs respectively) and provide graphics of the median.

We stress that no scan of the seed space was performed for ELRA, nor hyper-parameter tuning via validation data!

3.1 Mathematical 2D experiments

As proof of concept and to explore certain standard problems in gradient descent, we first show 2-dimensional results on saddle points, bowls/parabolas and the Rosenbrock function.

3.1.1 Saddle points

Saddle points (where $\nabla f(x) = 0$ but $f(x)$ is not a local max/min) can pose problems in gradient descent methods, as the gradient becomes arbitrarily small near them, which might lead to catastrophic speed loss. Generically, in suitable coordinates, these saddle points look locally like $x = (0, 0)$ for $f(x) = x_1^2 - x_2^2$ (see Math. Suppl., (4)). However, for a given data representation, it is more likely that the coordinates near a saddle are rotated!

We looked at the performance of the optimizers AdaDelta, Adam (with $\alpha = 0.01, \beta_1 = 0.9, \beta_2 = 0.999$), C2Min and P2Min near the standard saddle $f(x) = x_1^2 - x_2^2$ starting at[9] $x_0 = (1, 10^{-9})$ and the problem rotated by $45°$. Fig. 3 (Up) shows the value of f over steps t. The dashed lines belong to the rotated situation. The fastest are C2Min and P2Min, which have the same graph with or without rotation (invariance). AdaDelta and Adam are slower and suffer significantly from $45°$-rotation, as it makes the component wise modification of the Ada-family completely useless. Fig. 3 (Down) illustrates the paths in the x_1-x_2-plane chosen by the different optimizers. One sees that C2Min and P2Min follow fast the gradient direction, while the Ada-family either try to avoid the saddle directly (unrotated situation) or follow slowly the gradient direction. This shows one drawback of conditioning individual axis weights within the Ada-family. It illustrates also that the different optimizers often find different local/global minima. Noteworthy: C2Min (green) shows visible oscillations around the x_2-axis, which we use by design to decelerate.

3.1.2 Bowls and Rosenbrock

As a second class of mathematical experiments, we considered higher dimensional parabolas (so called bowls), i.e. functions of the form $f(x) = \sum_i c_i \cdot x_i^2$, and the infamous Rosenbrock function $f(x) = (1-x_1)^2+100(x_2-x_1^2)^2$. Bowls provide the simplest non-trivial functions for convex optimization, while the Rosenbrock function with its curved valley is a difficult standard optimization problem. Here, we used for Adam $\alpha = 0.05, \beta_1 = 0.8, \beta_2 = 0.9$ and for RMSprop $\alpha = 0.05$.

Table 1. *Steps t to reach $f(x_t) < \varepsilon$ from $x_0 = (-5.75, 1.75)$ for the bowl $f(x) = 3x_1^2 + 24x_2^2$*

accuracy	Adam	RMSprop	C2Min	P2Min
$\varepsilon = 10^{-1}$	128	142	20	**9**
$\varepsilon = 10^{-6}$	184	∞	42	**12**

[9] All true gradient descent methods fail when starting at $(1, 0)$.

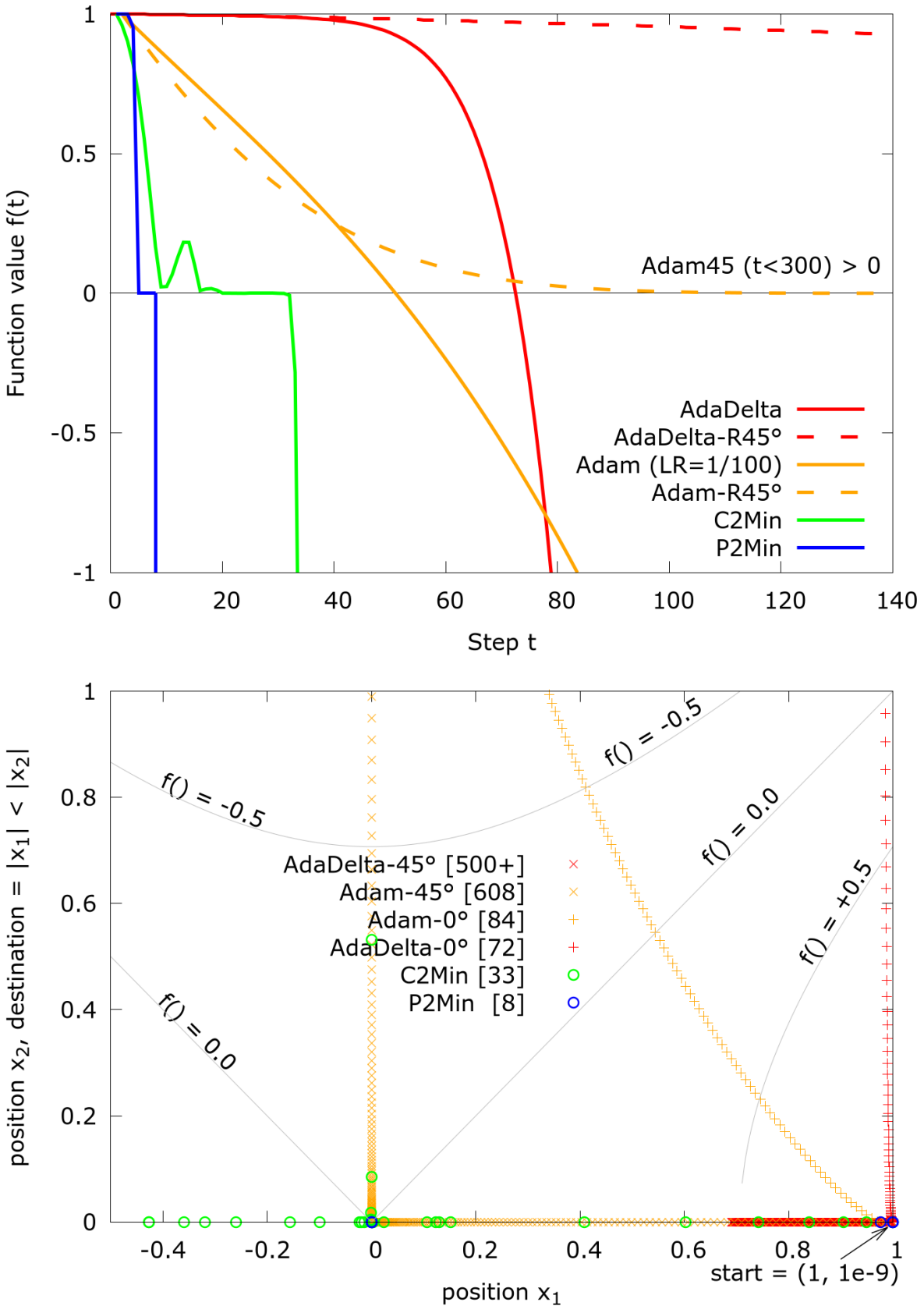


Figure 3. *Behaviour of optimizers near saddle $f(x) = x_1^2 - x_2^2$ and effect of $45°$-rotation. Up: steps t needed to reach < -1, Down: optimization paths (+ $\simeq 0°$, × $\simeq 45°$) in x_1-x_2-plane. Note: 4 of the 8 steps of P2Min (blue) are indistinguishable near $(0, 0)$.*

Table 2. *Steps t to reach $f(x_t) < 1$ from start point x_0 for Rosenbrock $f(x) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$*

start point	Adam	RMSprop	C2Min	P2Min
$(-3, -2)$	208	176	227	**10**
$(-11, 121)$	$> 10^4$	$> 10^4$	$> 10^4$	**300**

The Tables 1 and 2 give the minimal number of steps t needed for the different optimizers to reach a certain threshold for $f(x_t)$. One sees that for these examples (together with the saddle from above) P2Min is by far the fastest and for Rosenbrock with bigger starting points, it is the only optimizer that produces any meaningful results. C2Min's bad performance for Rosenbrock might be due to non-optimal momentum control. We hope to improve this result in the near future (see Future work B).

3.2 Neural networks

We conducted 7 different experiments (each for 10 initializations/seeds) with neural networks for image classification, involving the 5 training data sets MNIST, Fashion-MNIST, CIFAR-10, CIFAR-100 and Tiny ImageNet and 6 different neural networks ranging from tiny ($\sim$ 8k parameters) to substantial ($\sim$ 36 mil. parameters). We use varying data augmentations and batch shuffling after each epoch.

All experiments are conducted for the ELRA-optimizers C2Min, P2Min and the state-of-the-art optimizers Adam and Lion for comparison and we plot the median of each performance metric (train-loss[10], test-loss, test-accuracy). All optimizers have fixed hyperparameters for all experiments, with the batch size being the only exception! The batch size for Lion and Adam is $b = 256$ and the learning rates are $\alpha = 10^{-3}$ for Adam and $\alpha = 10^{-4}$ for Lion. The minimal batch sizes b_{min} (see section 2.4) are $b_{min} = 24$ for C2Min, $b_{min} = 32$ for P2Min for problems with 10 classes (MNIST, Fashion-MNIST, CIFAR-10) and $b_{min} = 256$ for C2Min, $b_{min} = 48$ for P2Min for problems with at least 1000 classes (CIFAR-100, Tiny ImageNet).

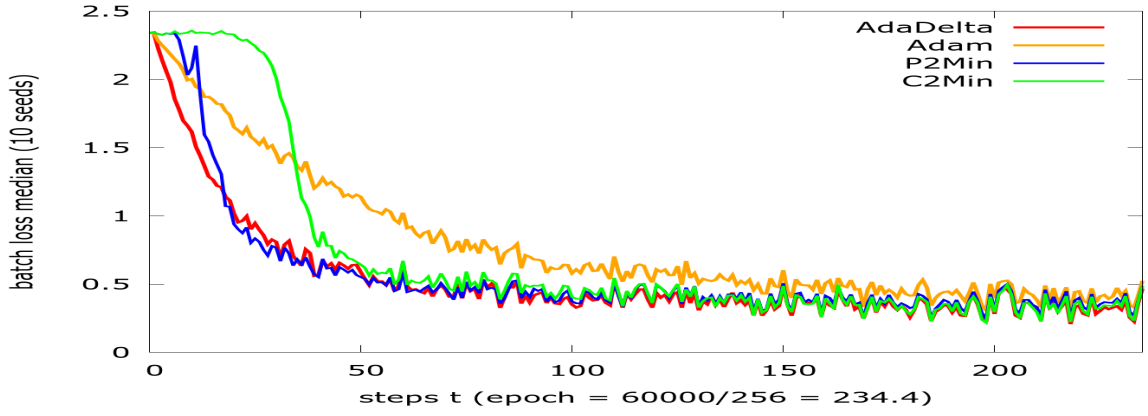


Figure 4. *Median Batch-Train-Loss for MNIST over 1^{st} epoch.*

Before presenting the experiments, we show in Fig. 4 the behaviour of C2Min and P2Min in the first epoch. The initial plateau phase for C2Min comes from the small initial learning rate $\alpha_0 = 10^{-5}$, requiring a fixed amount of steps to increase α to the right

[10]Train-loss is computed on all training data after each epoch.

magnitude, due to exponential adaptation. Note that P2Min has a significantly shorter adaption phase.

3.2.1 MNIST and Fashion-MNIST

We conducted experiments on the MNIST and Fashion-MNIST data set containing each $(60+10)$k pictures (28×28 pixels, gray-scale) of handwritten single digits (for MNIST) or fashion items of 10 different classes (for Fashion-MNIST). We used a primitive fully connected network with 1 hidden layer (10 neurons) and ReLU-activation for MNIST and the 3-layer convolutional network FashionCNN for Fashion-MNIST. No data augmentation was used for MNIST, while we utilized random horizontal flips for Fashion-MNIST. Each experiment was run for 80 epochs.

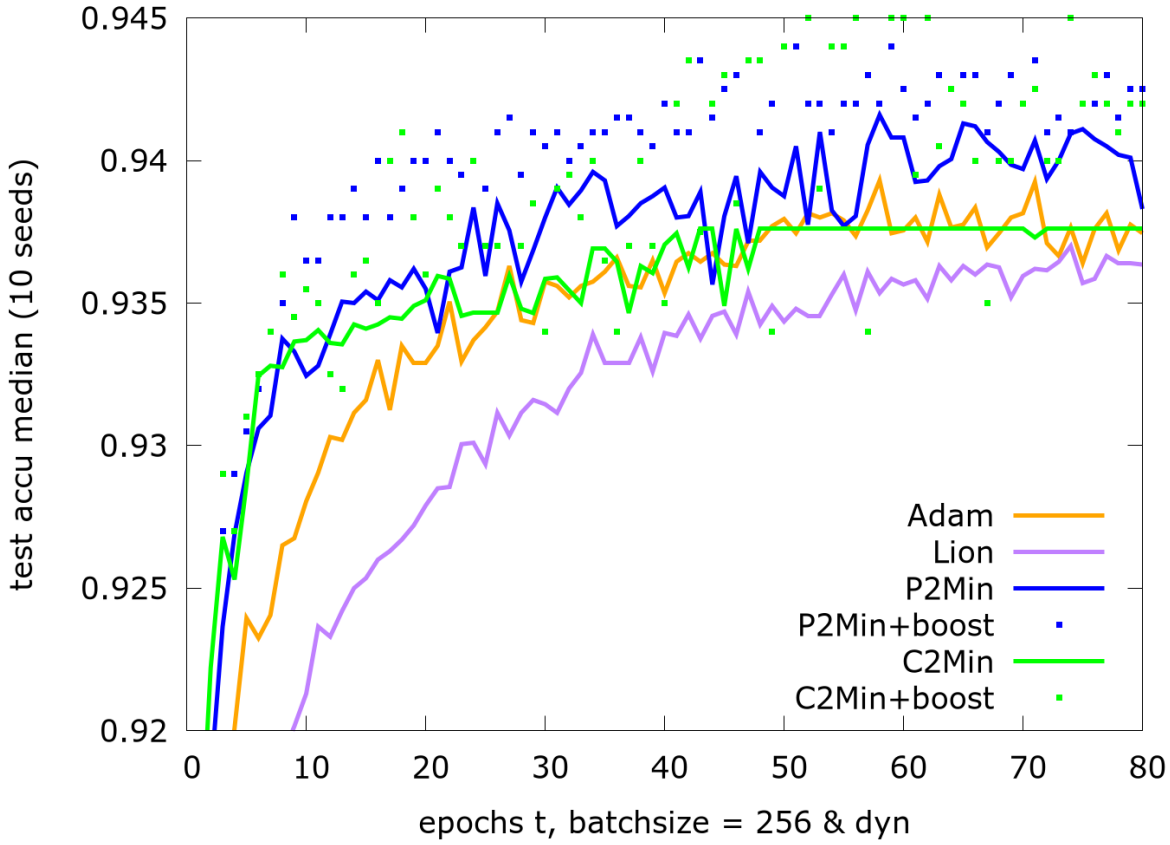


Figure 5. *Median Test-accuracy for MNIST for 80 epochs*

In Fig. 5 and Fig. 6, we show the median test accuracy. The dotted values are the possible results provided by the final boost after each epoch (see section 2.5). The test- and train-loss data can be found in Fig. 14a-15b in the appendix.

Particular remarkable is that for MNIST and Fashion-MNIST, C2Min and P2Min have significantly lower test-losses than Adam and Lion by comparable test-accuracy.

3.2.2 CIFAR-10

The CIFAR-10 data set contains $(50+10)$k images (32×32 pixels, RGB color) of objects of 10 different classes. We use the standard residual neural networks ResNet18, ResNet34, see [14], and Wide-ResNet-28-10, see [15]. For data augmentation, we use PyTorch's

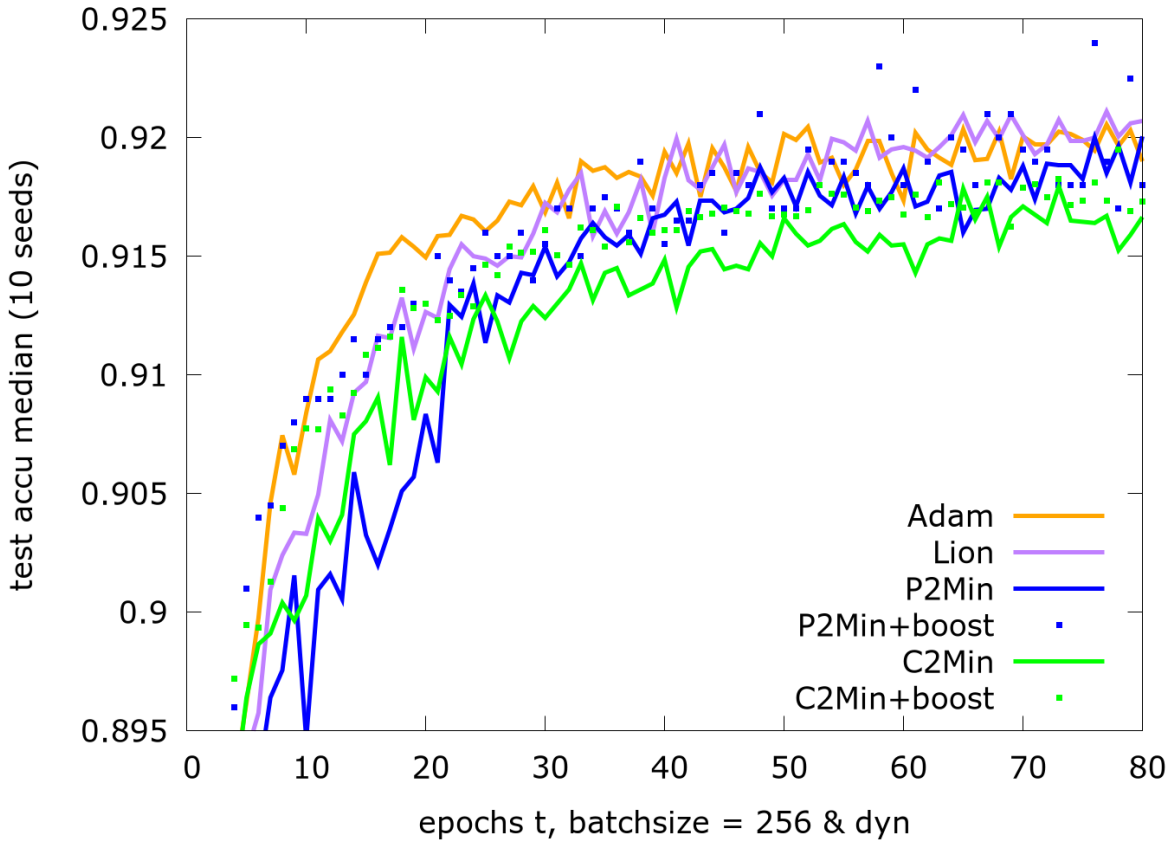


Figure 6. *Median Test-accuracy for Fashion-MNIST for 80 epochs*

RandomCrop with padding=4 in reflect mode, random horizontal flips and batch shuffling. Here, each experiment was run for 60 epochs. Note that we tested Wide-ResNet-28-10 only on 3 seeds each, due to computational costs.

We show in Fig. 7-10 the median test accuracies (see App., Fig. 16a-19b for test- and train-loss). The most striking observation from these experiments is the huge accuracy gap between Adam and Lion on one side and P2Min and C2Min on the other (see section 3.3 for possible explanations).

The steep increase of P2Min between epochs 35 and 40 in all these experiments comes from the fact that around this time this optimizer starts to increase the batch size. As a consequence, the oscillations around the optimal descent path (see section 2.5) are also drastically reduced and the benefit of boosting vanishes almost completely.

It is noteworthy that the test-loss obtained by P2Min is by far the smallest, although the train-loss is comparable to that of C2Min (and smaller than that of Adam and Lion). This suggests that P2Min reaches in fact a different local minimum than all the other optimizers.

As a final result, Fig. 10 shows the performance of C2Min and P2Min for ResNet18 with fixed (!) batch sizes 256 or 64 resp. (see Fig. 19a and Fig. 19b for test- and train-loss). This illustrates that the dynamically adapted learning rate alone is already a strong tool which leads to better performances than that of Lion or Adam. It shows also that P2Min benefits strongly from dynamic batch sizes, while C2Min has even a better accuracy result with the fixed batch size, suggesting that the batch size control is not yet optimal.

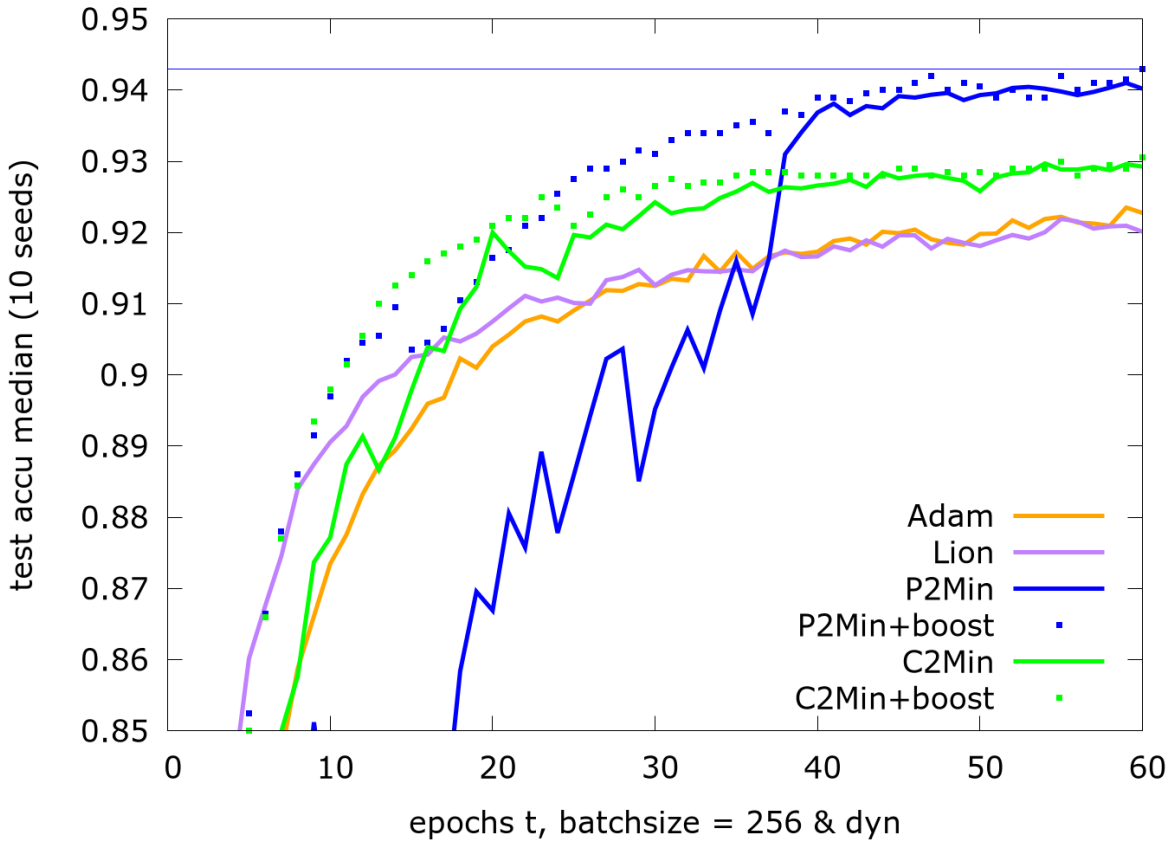


Figure 7. *CIFAR-10, ResNet18: Median Test-accuracy for 60 eps.*

3.2.3 CIFAR-100 and Tiny ImageNet

The CIFAR-100 data set is similar to CIFAR-10, containing $(50{+}10)$k images ($32{\times}32$ pixels, RGB color) of objects of 100 classes, so that each object is represented only on 500 images in the training set (compared to 5000 for CIFAR-10). This explains partly, why ResNet18, which we use here, performs much worse on CIFAR-100 than on CIFAR-10. As for CIFAR-10, the experiments run for 60 epochs and we use the same data augmentations, i.e. RandomCrop and random horizontal flips.

The final data set Tiny ImageNet contains $(95{+}5)$k images ($64{\times}64$ pixels, RGB color) of objects of 200 classes. Here, we trained ResNet50 (see [14]). As this is a downsized version of the much larger data set ImageNet, we resized the images to $256{\times}256$ pixels and then cropped 16 pixels in all directions. For data augmentation, we only used random horizontal flips. As these experiments are much larger (and consequently require much more computation time), we have only performed two runs, one for C2Min and one for P2Min, each running over 60 epochs.

Fig. 11-12 show the median test accuracies (see App., Fig. 20a-21b for test- and train-loss). For CIFAR-100, the test-accuracy behaves similar to CIFAR-10, but the gap between Lion/Adam and C2Min/P2Min is even bigger (note the different scaling). Again, the effect of the increased batch size on P2Min is clearly visible around the 30^{th} epoch.

For Tiny ImageNet, we included in Fig. 12 for comparison test/validation accuracies obtained by other authors (see also Fig. 13). The better result of C2Min could be chance (only 1 seed!).

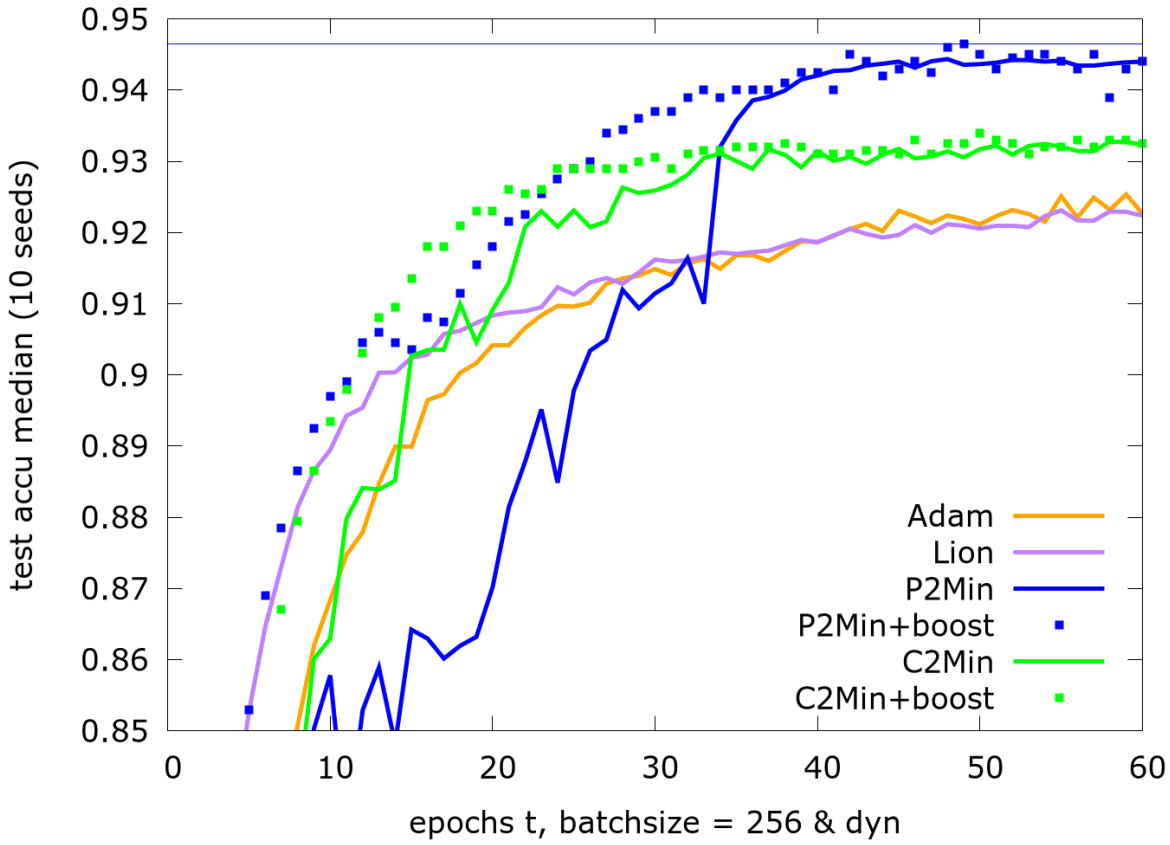


Figure 8. *CIFAR-10, ResNet34: Median Test-accuracy for 60 eps.*

3.3 Analysis

While for the almost trivial problems MNIST and Fashion-MNIST, advantage of the ELRA optimizers over whose from the Ada-family is only marginal, it is overwhelming for CIFAR. This might stem from the fact that ELRA is closer to the generic stochastic gradient descent (SGD) optimizer, which corresponds to setting $\alpha = const.$ and $\beta = 0$ in the update scheme (1). For SGD it has been observed that, albeit being very slow, it tends to yield minimizers, which generalize better to the test data. This is explained by SGD having noisier optimization paths, which helps to escape steep local minima, which generalize less optimal. See [19] and [20] for some explanations of this effect. This means that our algorithms combine the best of two worlds: the speed of Adam/Lion with the better generalization of SGD.

Moreover, dynamical adaption of the batch size has the same effect as a learning rate schedule, but much faster, illustrated by Fig. 13, taken from [21] and [18], where we included for comparison our results from Fig. 9 and Fig. 12.

The advantage of P2Min over C2Min on all CIFAR experiments comes probably from the fact that P2Min uses no momentum. Momentum can help to gain speed and optimization path stability at the beginning, yet hampers increasingly the progress close to the minimum until it forces x essentially to rotate around the critical point, unless α, β or δ are reduced.

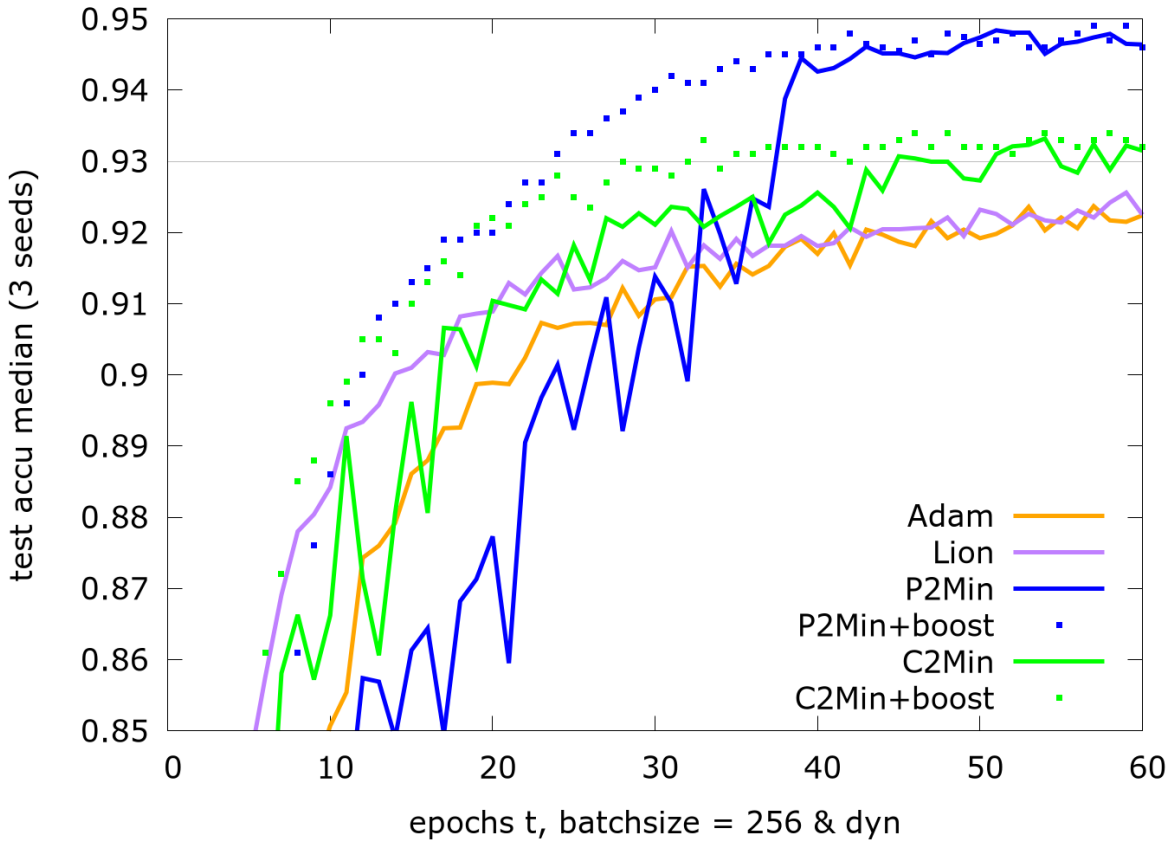


Figure 9. *CIFAR-10, Wide-ResNet-28-10: Median Test-accuracy for 60 epochs*

4. Limitations

Our implementations still leave much room for improvements. For instance (with very small batch sizes), C2Min and P2Min can fail by increasing α over a long period, thereby increasing the noise and finally becoming unstable. This could be prevented by more conservative α updates, yet these can lead to speed loss and reduced performance. Moreover, the controls for momentum and batch size are functional, yet far from optimal. Finding a more universal solution here requires more research.

Finally, our code and consequently the computational resources would benefit from ELRA specific adaptations of the PyTorch or Jax architecture, such as providing by default the function value $f(x_t)$ and the scalar products $\langle G_t, G_t\rangle$, $\langle G_{t-1}, G_t\rangle$ to the optimizer and a flexible data loader for varying batch sizes.

5. Conclusion

We presented a novel, simple, mainly self consistent, robust and fast optimizing method with linear dimensional scaling and rotational invariance, realized in two algorithms. Typical runs on mathematical standard problems and statistical tests on neural networks for the MNIST, CIFAR and Tiny ImageNet data sets with several initializations showed better final test losses then the best state of the art optimizers Adam or Lion with hand-tuned optimal parameters!

We believe that nobody has thought about trying steep and fast α-adaptions before due to the following reasons: for small dimensions good solvers exist (often using matrix inversions, e.g. the Levenberg–Marquardt algorithm), mathematical optimizers strive

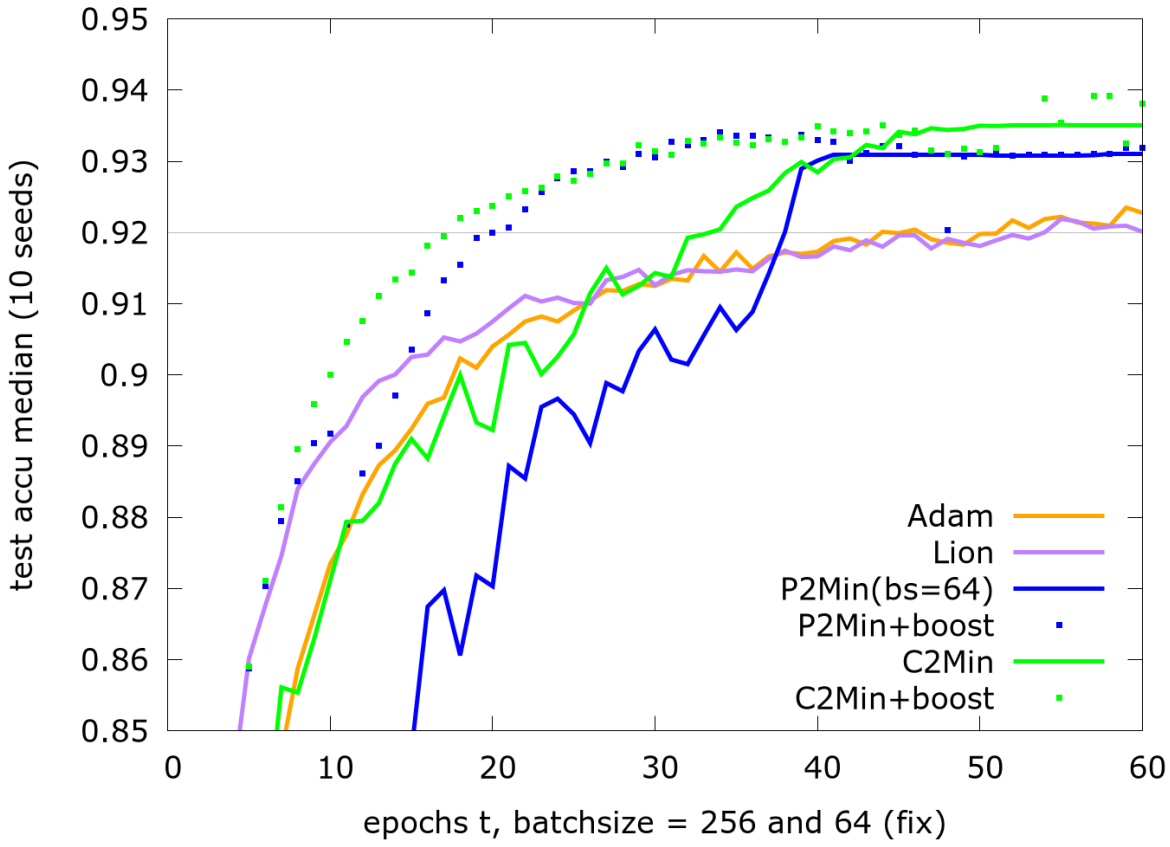


Figure 10. *CIFAR-10, ResNet18: Median Test-accuracy for 60 epochs, with constant batch-size 256*

for provability (which restricted until recently to constant α: compare [7] and [8]) and previous conditions (Armijo) for updating α are too expensive in high dimensions.

Additionally, better control systems for learning rate, momentum and soft restarts promise increased performance and universality (see Future works below). Meta-learning, e.g. applying a small neural network to fine-tune the control parameters, could also lead to further improvement.

We strongly believe that the above ideas will create a completely new research field in gradient descent-based optimization.

Author contributions

A.K. and A.F. conceived of the presented idea, and both developed the theory and performed the computations. A.K. and A.F and S.K. verified the analytical methods. All authors discussed the results and contributed to the final manuscript.

Competing interests

The authors declare that they have no competing interests.

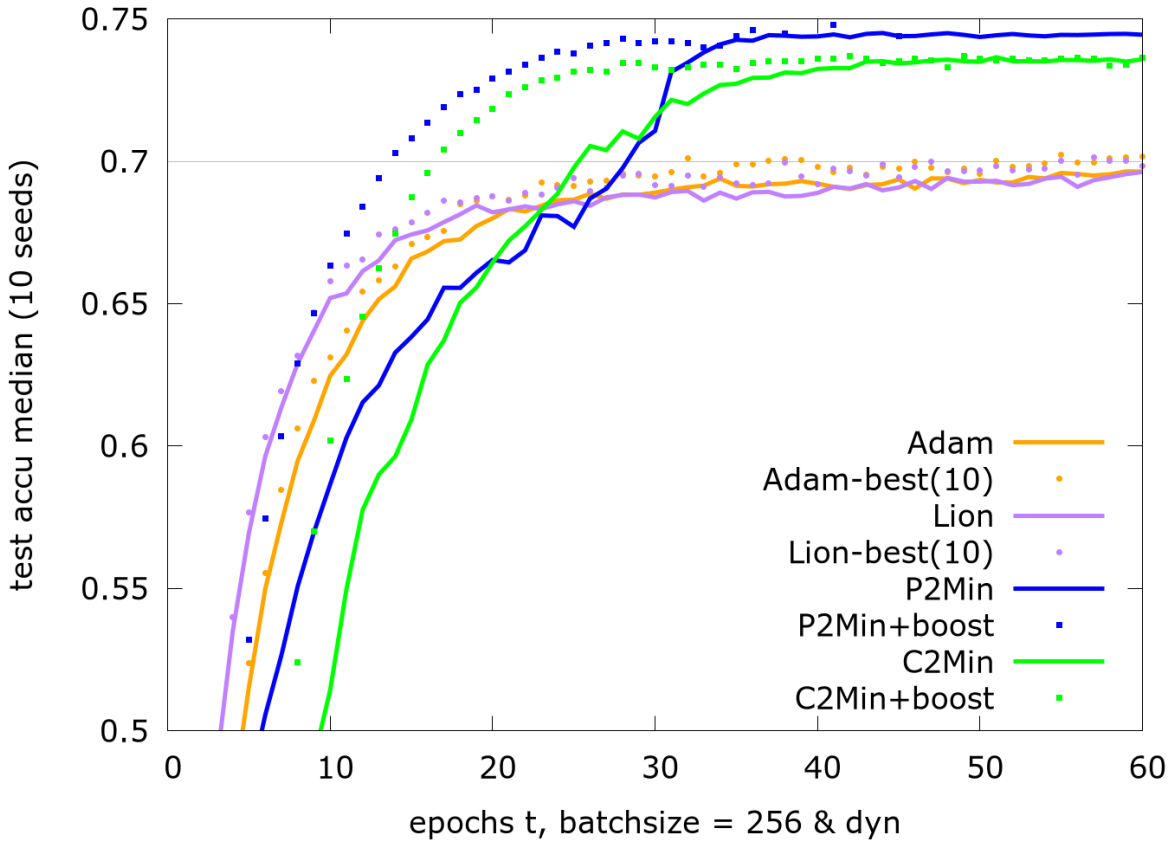


Figure 11. *CIFAR-100, ResNet18: Median Test-accu. for 60 eps.*

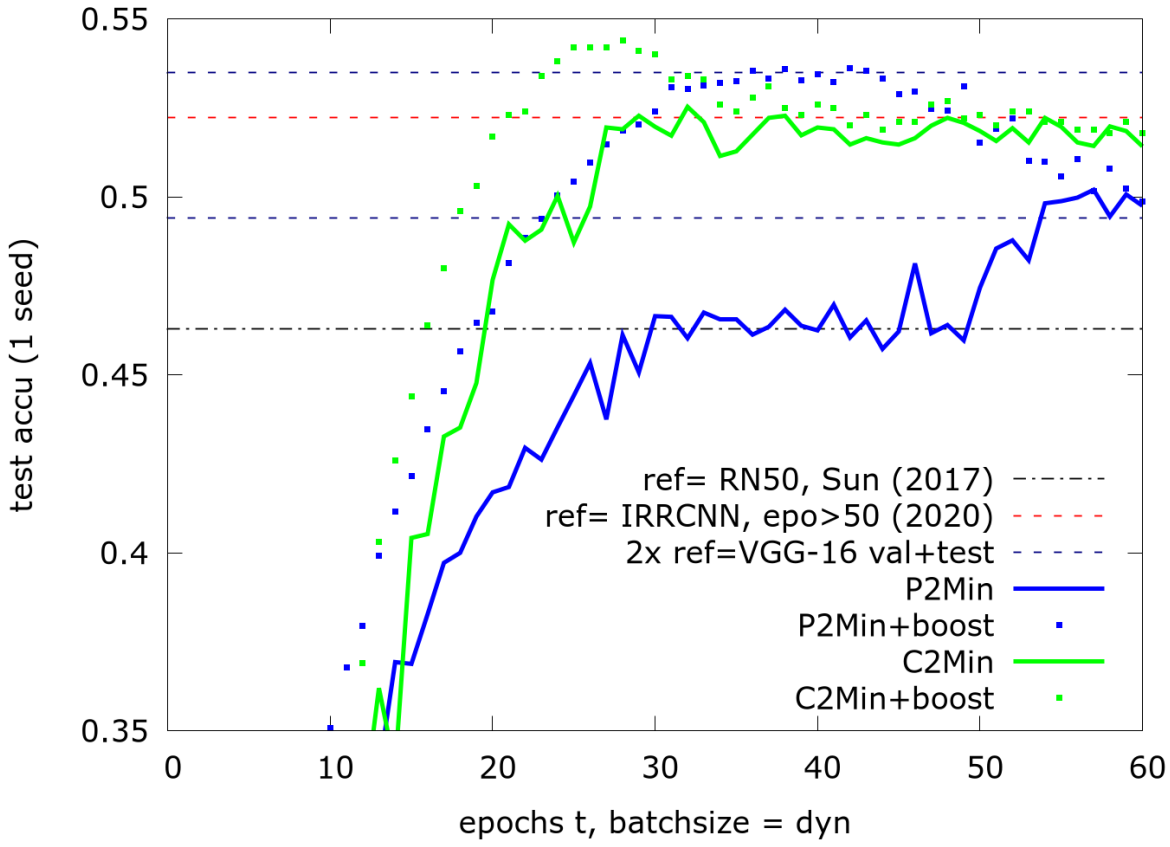
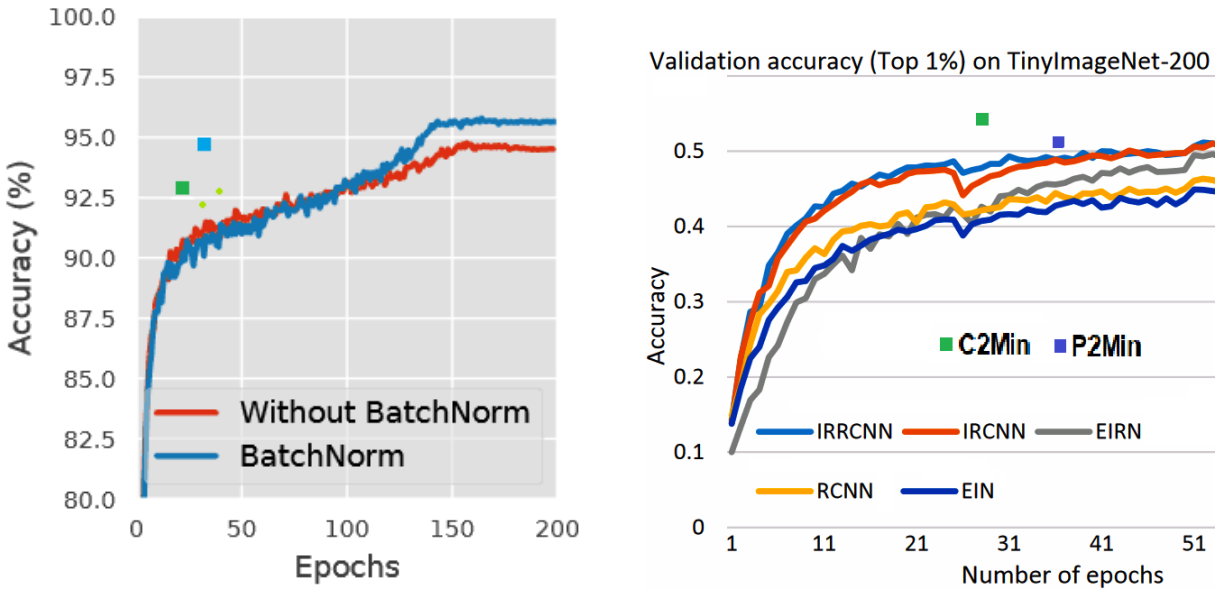


Figure 12. *Tiny ImageNet, ResNet50: Median Test-accuracy for 60 epochs, reference lines [16], [17], [18]*



(a) *CIFAR-10* ***(b)*** *Tiny ImageNet*

Figure 13. *Test-Accuracy for (a) CIFAR-10 with Wide-ResNet-28-10 from [21] and (b) Tiny ImageNet with ResNet50 from [18]. The green/blue dots are our results (cf. Fig. 9, 12). Note that we need only roughly* 25% *in (a) and* 50% *in (b) of the time to obtain the same final accuracy.*

References

[1] O. Borysenko and M. Byshkin, "Coolmomentum: A method for stochastic optimization by langevin dynamics with simulated annealing", *Scientific Reports*, vol. 11, p. 10 705, 2021. DOI: 10.1038/s41598-021-90144-3.

[2] M. N. Ab Wahab, S. Nefti-Meziani, and A. Atyabi, "A comprehensive review of swarm optimization algorithms", *PloS one*, vol. 10, no. 5, e0122827, 2015.

[3] K. Mishchenko and A. Defazio, *Prodigy: An expeditiously adaptive parameter-free learner*, 2023. arXiv: 2306.06101 [cs.LG].

[4] M. Ivgi, O. Hinder, and Y. Carmon, *Dog is sgd's best friend: A parameter-free dynamic step size schedule*, 2023. arXiv: 2302.12022 [cs.LG].

[5] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[6] X. Chen et al., "Symbolic discovery of optimization algorithms", *ArXiv*, vol. abs/2302.06675, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:256846990.

[7] Y. Nesterov, *Lectures on Convex Optimization*, 2nd. Springer Publishing Company, Incorporated, 2018, ISBN: 3319915770.

[8] B. Grimmer, *Provably faster gradient descent via long steps*, 2023. arXiv: 2307.06324 [math.OC].

[9] T. T. Truong and H.-T. Nguyen, "Backtracking gradient descent method and some applications in large scale optimisation. part 2: Algorithms and experiments", *Applied Mathematics & Optimization*, vol. 84, pp. 2557–2586, 2021. DOI: 10.1007/s00245-020-09718-8. [Online]. Available: https://doi.org/10.1007/s00245-020-09718-8.

[10] S. Ling, N. Sharp, and A. Jacobson, *Vectoradam for rotation equivariant geometry optimization*, 2022. DOI: 10.48550/ARXIV.2205.13599. [Online]. Available: https://arxiv.org/abs/2205.13599.

[11] A. Kleinsorge, S. Kupper, A. Fauck, and F. Rothe, *Elra: Exponential learning rate adaption gradient descent optimization method*, 2023. arXiv: 2309.06274 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2309.06274.

[12] anonymous git, *Python elra solver in git*, 2024. [Online]. Available: https://anonymous.4open.science/r/solver-347A/README.md.

[13] S. L. Smith, P. Kindermans, and Q. V. Le, "Don't decay the learning rate, increase the batch size", *CoRR*, vol. abs/1711.00489, 2017. arXiv: 1711.00489. [Online]. Available: http://arxiv.org/abs/1711.00489.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *CVPR2016*, Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[15] S. Zagoruyko and N. Komodakis, *Wide residual networks*, 2017. arXiv: 1605.07146 [cs.CV].

[16] L. Sun, "Resnet on tiny imagenet", 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:196590979.

[17] A. Shcherbina, *Tiny imagenet challenge*, 2016. [Online]. Available: http://cs231n.stanford.edu/reports/2016/pdfs/401_Report.pdf.

[18] M. Z. Alom, M. Hasan, C. Yakopcic, T. M. Taha, and V. K. Asari, "Improved inception-residual convolutional neural network for object recognition", *Neural Computing and Applications*, vol. 32, pp. 279–293, 2020. DOI: 10.1007/s00521-018-3627-6. [Online]. Available: https://doi.org/10.1007/s00521-018-3627-6.

[19] P. Zhou, J. Feng, C. Ma, C. Xiong, S. Hoi, and E. Weinan, "Towards theoretically understanding why sgd generalizes better than adam in deep learning", in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS'20, Vancouver, BC, Canada: Curran Associates Inc., 2020, ISBN: 9781713829546.

[20] W. R. Huang et al., "Understanding generalization through visualizations", *CoRR*, vol. abs/1906.03291, 2019. arXiv: 1906.03291. [Online]. Available: http://arxiv.org/abs/1906.03291.

[21] Y. Dauphin and E. D. Cubuk, “Deconstructing the regularization of batchnorm”, in *International Conference on Learning Representations*, 2021. [Online]. Available: https://openreview.net/forum?id=d-XzF81Wg1.

[22] J. Milnor, *Lectures on the H-Cobordism Theorem*. Princeton: Princeton University Press, 1965, ISBN: 9781400878055. DOI: doi:10.1515/9781400878055. [Online]. Available: https://doi.org/10.1515/9781400878055.

A. Methods details

A.1 Summary

Table 3. *Properties of Adaptive Optimizer Features*

Main feature	Level of maturity	General applicable
dynamic alpha	great (fast, robust, universal)	yes, simple
dynamic beta	working, but poorly motivated	*could be, not tried*
dynamic batch-size	good (fast, robust, universal)	yes, independent of any SGD optimizer
(final) boosting	great, improves most results	could help Adam (etc.) in final epochs
soft restart	good, reverts failed step	simple, more robustness with high α

A.2 Estimating α using a parabola ansatz

To get the update formula for α of P2Min, we consider f only along the straight line through x_{t-1} and x_t, whose direction is $x_t - x_{t-1} = -\alpha_{t-1} G_{t-1}$. We assume that f along this line is a parabola, i.e. $f(x) = ax^2 + b$, where we chose (in practice unknown) coordinates such that $x = 0$ is the minimizer of f. In this setting, the derivatives of f are:

$$2ax_{t-1} = f'(x_{t-1}) = \partial_{G_{t-1}} f(x_{t-1}) = ||G_{t-1}|| \quad \text{and} \quad 2ax_t = f'(x_t) = \partial_{G_{t-1}} f(x_t) = \frac{\langle G_{t-1}, G_t \rangle}{||G_{t-1}||},$$

where $\partial_{G_{t-1}}$ is the directional derivative of f with respect to G_{t-1}. Together with $x_t - x_{t-1} = -\alpha_{t-1} G_{t-1}$, we get

$$2a(x_t - x_{t-1}) = \frac{\langle G_{t-1}, G_t \rangle}{||G_{t-1}||} - ||G_{t-1}|| = \frac{\langle G_{t-1}, G_t \rangle - ||G_{t-1}||^2}{||G_{t-1}||} \quad \Longrightarrow \quad a = \frac{\langle G_{t-1}, G_t \rangle - ||G_{t-1}||^2}{-2\alpha_{t-1} \cdot ||G_{t-1}||^2}$$

As we want $x_t - \alpha_t \cdot f'(x_t) = x_{t+1} = 0$ to be the minimizer of f, we obtain with $x_t = \frac{f'(x_t)}{2a}$ that

$$\begin{aligned} \alpha_t = \frac{x_t}{f'(x_t)} = \frac{1}{2a} = \alpha_{t-1} \cdot \frac{||G_{t-1}||^2}{||G_{t-1}||^2 - \langle G_{t-1}, G_t \rangle} &= \alpha_{t-1} \cdot \left(1 + \frac{\langle G_{t-1}, G_t \rangle}{||G_{t-1}||^2 - \langle G_{t-1}, G_t \rangle} \right) \\ &= \alpha_{t-1} \cdot \left(1 + \frac{\cos_t}{||G_{t-1}||/||G_t|| - \cos_t} \right), \end{aligned} \tag{3}$$

where we used $\langle G_{t-1}, G_t \rangle = \cos_t \cdot ||G_{t-1}|| \cdot ||G_t||$ in the final step.

A.3 Noise correction multiplier κ for P2Min

When training neural networks, the update scheme (3) for α in P2Min estimates α too small. We believe that this is due to stochastic noise in the gradients G_t, coming from the use of data batches much smaller than the full data set. To rectify this behaviour, we introduce a corrective multiplier κ, such that $\alpha_t = \alpha_{t-1} \cdot \left(1 + \frac{\cos_t}{||G_{t-1}||/||G_t|| - \cos_t}\right) \cdot \kappa$. The multiplier is calculated by the formula:

$$\frac{1}{\kappa} = 1 - \min \left(0.1 \, , 0.14 \cdot \left(\frac{2.35^2}{f(x_0)^2} \cdot f(x_{best}) \right)^{0.8} \right).$$

Here, $f(x_{best})$ is a moving average over the best function values (see below A.4). The value $f(x_0)$ is the initial value of f. The term $\frac{2.35^2}{f(x_0)^2}$ is a dimensional correction term, which compensates the fact that for networks with more output classes, $f(x_{best})$ is higher. The constant 2.35 is the typical initial value $f(x_0)$ for networks with 10 classes, such

as for MNIST or CIFAR-10. We expect that $f(x_0)$ could be replaced by the stochastic expected value of f, if the network is initialized randomly. Note that we assume for this κ that the loss function satisfies $f(x) > 0$ away from the global minimum.

A.4 Averaging the best function values

At several control points, such as soft restarts and the above presented corrective multiplier κ, we use an average $f(x_{best})$ over the best values of f. It is recursively defined as follows:

$$f(x_{best}) = \begin{cases} f(x_t) & \text{if } f(x_t) < f(x_{best}) \\ 1.1 \cdot f(x_{best}) & \text{otherwise} \end{cases}.$$

This definition gives roughly the best value of f, but slowly moves away from it, if no better value is seen. This guarantees that one is not stuck with a very small $f(x_{best})$, which was only obtained by chance. Note that $1.1 \cdot f(x_{best})$ only increases $f(x_{best})$ if we have always $f(x) > 0$ away from the global minimum.

A.5 Momentum control

As explained above (see section 2.3), we use a typical momentum update scheme $M_t = \delta \cdot M_{t-1} + (1-\delta) \cdot G_{t-1}$ with $\delta = 0.8$, but a dynamically controlled different parameter β_t for the actual step $x_{t+1} = x_t - \alpha_t\big((1-\beta_t)G_t + \beta_t M_t\big)$. We control β_t by the average relative oscillation $\overline{o}$ of $f(x)$ around the mean function value $\overline{f}$ given as follows:

$$\begin{aligned}
\overline{f}_t &= 0.9 \cdot \overline{f}_{t-1} + 0.1 \cdot f(x_{t-1}) && \text{(moving mean of } f\text{)}, \\
o_t &= \max\left(0.043, \min\left(0.7, \left|\frac{f(x_{t-1}) - \overline{f}_t}{\overline{f}_t}\right|\right)\right) && \text{(oscillation at } x\text{, truncated to } [0.043, 0.7]\text{)} \\
\overline{o}_t &= 0.9 \cdot \overline{o}_t + 0.1 \cdot o_t && \text{(moving mean of oscillations)} \\
\beta_t &= 0.85 - 1.2 \cdot \overline{o}_t
\end{aligned}$$

The truncation of o_t to the interval $[0.043, 0.7]$ guarantees that the momentum control parameter β_t lies between 0.01 and $0.7984 < \delta = 0.8$. Here, larger oscillations lead to less momentum, while small oscillations give more. The idea is that a large momentum in a landscape with high curvature can induce additional oscillations and consequently slow down the optimization. However, this control was designed purely phenomenological (by try and error) and relies on the fact, that for neural networks, the loss function $f(x) > 0$ away from the global minimum, so that the fraction in o_t is well-defined. Ideally, $\overline{f}$ should be replaced by the mean distance to the optimal x, yet this is in practice unknown. Hence, controlling the momentum leaves much room for future improvement.

A.6 Dynamical batch size control

As explained above (see section 2.4), we control the batch size $b_t = m_t \cdot b_{min}$ by an integer multiplier m_t, which is updated every 100 steps using the mean $\overline{s}$ of the cosines

of deviation angles for the gradients G_t. The explicit update formula is:

$$k_t = \begin{cases} (0.015 - \overline{s}) \cdot 25 + 1 & \text{for } \overline{s} \leq 0.015 \\ 1 & \text{for } 0.015 < \overline{s} \leq 0.025 \\ (0.025 - \overline{s}) \cdot 12.5 + 1 & \text{for } \overline{s} > 0.025 \end{cases}$$

$$m_t = \begin{cases} \max\left(2, \lfloor m_t \cdot k_t \rfloor\right) + 2 & \text{if } s \leq 0.015 \\ \max\left(2, \lfloor m_t \cdot k_t \rfloor\right) & \text{otherwise} \end{cases}$$

This formula guarantees that m_t increases by at least 2, if $s \leq 0.015$ (needed as $m_t = \lfloor m_t \cdot k_t \rfloor$ if $m_t \cdot k_t < m_t + 1$). Moreover, it increases the batch size faster, than decreasing it.

B. Future work

Opening a new field creates lots of opportunities for continuation. Here we mention some of the most promising directions:

- $\alpha = \alpha \cdot (1 + \cos \cdot g(x))$ is the general update scheme for α obtained from our idea of orthogonal gradients (2). Here, g can be any function with $g(x) > 0$. What is the best g? Different answers for different problems?
- Short initial phase (warm-up) to find faster ideal initial α_0 and ideal minimal batch size b_{min}
- Problem specific fine tuning (selected hyper-parameters) is possible and could give further improvement:
  - fixed bounds for α (i.e. $10^{-7} < \alpha < 10^{-1}$) based on statistics gathered during current run. Could speed up P2Min significantly (fewer soft restarts, shorter time to recover from restart)
  - better dynamic adaption of momentum parameters β, δ for C2Min
- Further applications: electronic structure optimizations, protein folding, molecular dynamics, finite element methods
- Possible landscape characterization as a side-result

C. Mathematical supplements

C.1 Extremal points sit inside quadratic surrounding

In principle, critical points x_0, such as local/global minima and saddle points, can be degenerate, i.e. the Hessian at x_0 can have 0 as an eigenvalue. However, functions with all critical points non-degenerate, so called Morse functions, are the generic situation, meaning that they form an open and dense subset within $C^2(\mathbb{R}^n)$, see [22]. So figuratively speaking, "almost all" two times continuously differentiable functions have only non-degenerate critical points. For these functions f, we find then by Taylor expansion, that they are locally dominated by their Hessian, i.e. they behave locally around critical points like quadratic functions:

$$f(x) = \sum_{i=1}^{n} c_i \cdot x_i^2, \qquad c_i \in \{+1, -1\}. \tag{4}$$

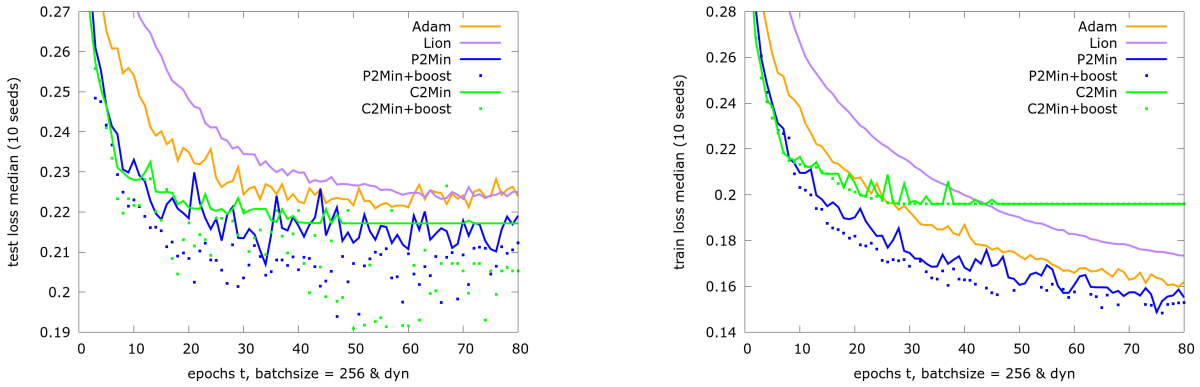
C.2 Random noise convolution removes discontinuities in Gradients

In some applications, the function f, which we want to optimize, is not differentiable, such as $f(x) = ||x||$ or $f(x) = \max\{x, 0\}$. Then, the gradient is not everywhere defined and most optimization methods suffer. However, if the data contains some random noise, i.e. the function f is slightly blurred, then we can expect differentiability. Indeed, the effect of random noise can be thought of as convoluting f with a probability density function ϕ, such as the density of the normal distribution $\phi(x) = exp(-x^2/2\sigma^2)/(\sigma\sqrt{\pi})$ (if the blurring can be arbitrarily large) or a density with finite support, if the blurring is limited. Now, if ϕ is continuously differentiable and f integrable or locally integrable (for finite support), then it is a well known fact that the convolution $f * \phi$ is also differentiable with differential

$$\partial_{x_i}(f * \phi)(x) = \partial_{x_i} \int_{\mathbb{R}^n} f(t) \cdot \phi(x - t)\mathrm{d}t = \int_{\mathbb{R}^n} f(t) \cdot \partial_{x_i}\phi(x - t)\mathrm{d}t = (f * \partial_{x_i}\phi)(x). \quad (5)$$

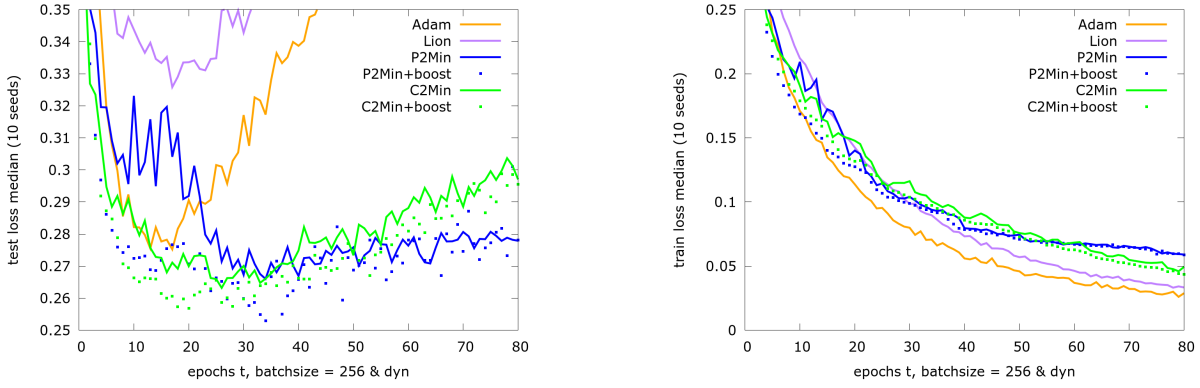
Especially DNN learning should be affected by noise from the input and from batching, resulting in smooth landscapes.

D. Additional performance plots



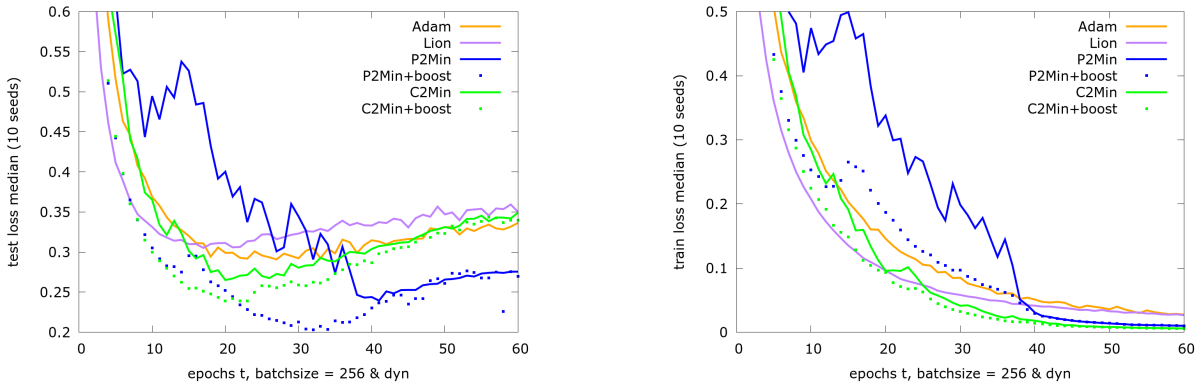
(a) *Test-loss* ***(b)*** *Train-loss*

Figure 14. *Median Test-/Train-loss over 80 epochs for MNIST, with learning rates $\alpha = 10^{-3}$ (Adam), $\alpha = 10^{-4}$ (Lion).*



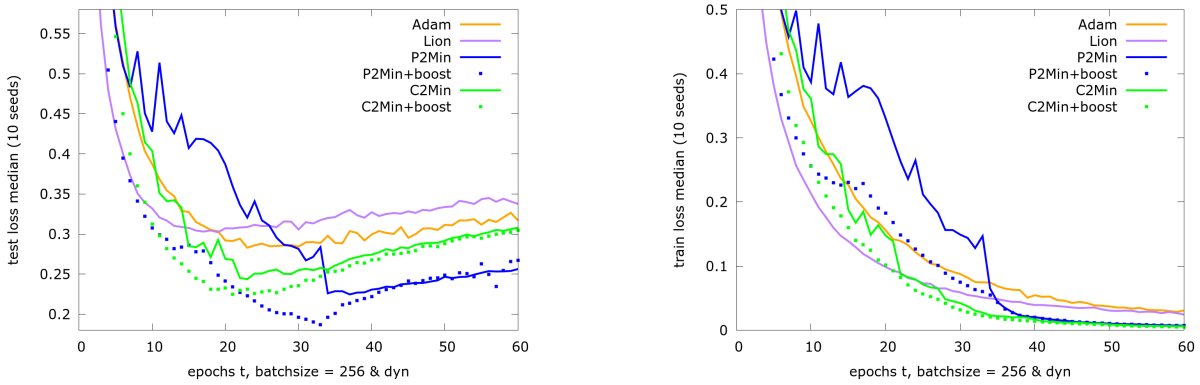
(a) *Test-loss* ***(b)*** *Train-loss*

Figure 15. *Median Test-/Train-loss over 80 epochs for Fashion-MNIST, with learning rates $\alpha = 10^{-3}$ (Adam), $\alpha = 10^{-4}$ (Lion).*



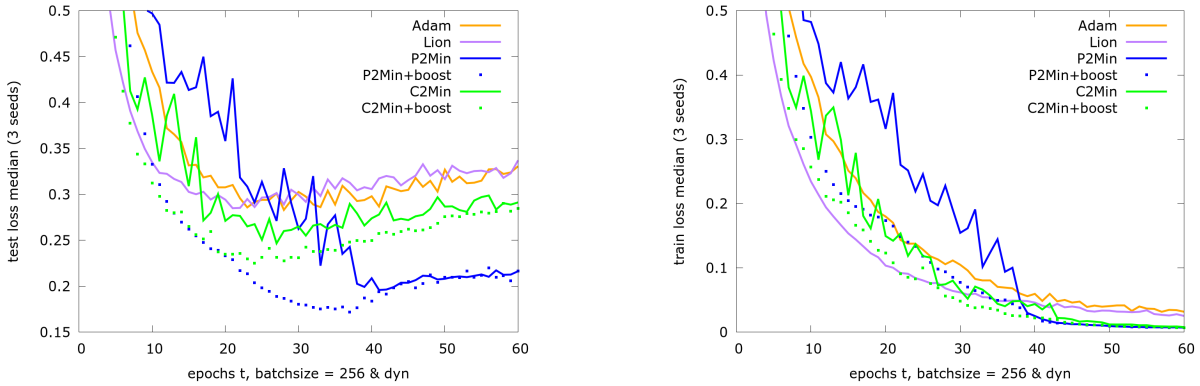
(a) *Test-loss* ***(b)*** *Train-loss*

Figure 16. *Median Test-/Train-loss over 60 epochs for CIFAR-10 using ResNet18, with* $\alpha = 10^{-3}$ *(Adam),* $\alpha = 10^{-4}$ *(Lion).*



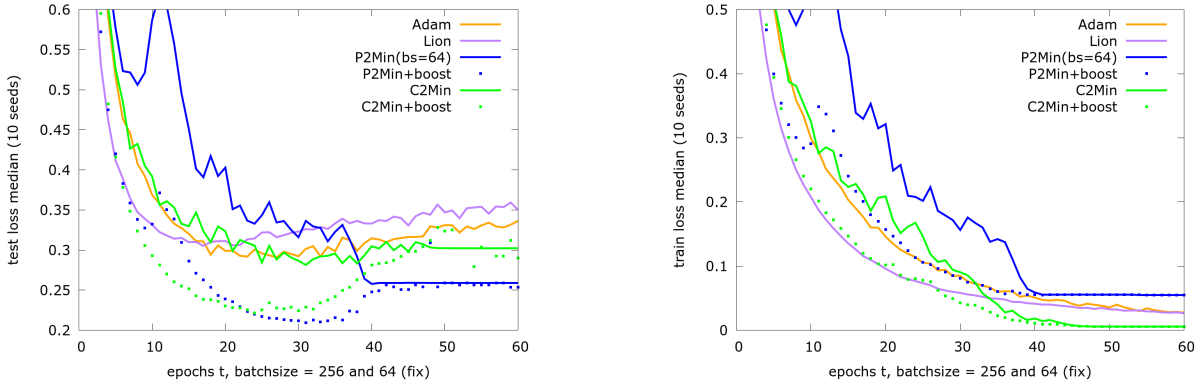
(a) *Test-loss* ***(b)*** *Train-loss*

Figure 17. *Median Test-/Train-loss over 60 epochs for CIFAR-10 using ResNet34, with* $\alpha = 10^{-3}$ *(Adam),* $\alpha = 10^{-4}$ *(Lion).*



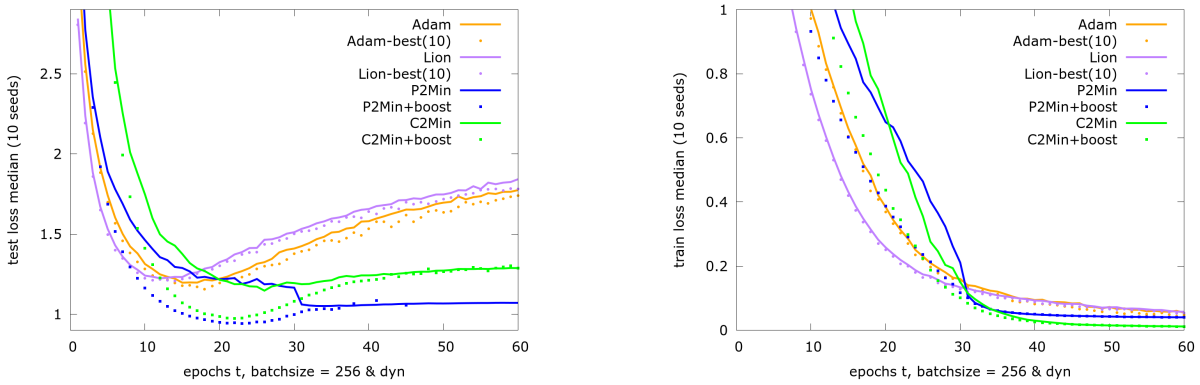
(a) *Test-loss* ***(b)*** *Train-loss*

Figure 18. *Median Test-/Train-loss over 60 eps. for CIFAR-10 on Wide-ResNet-28-10, with* $\alpha = 10^{-3}$ *(Adam),* $\alpha = 10^{-4}$ *(Lion).*



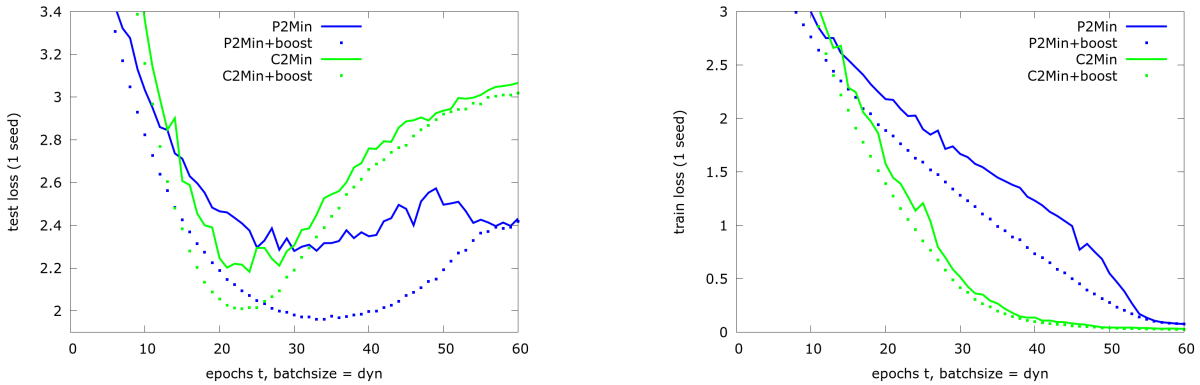
(a) *Test-loss* ***(b)*** *Train-loss*

Figure 19. *Median Test-/Train-loss over 60 eps. for CIFAR-10 on ResNet18, with* $b = 256$ *(fixed),* $\alpha = 10^{-3}$ *(Adam),* $\alpha = 10^{-4}$ *(Lion).*



(a) *Test-loss* ***(b)*** *Train-loss*

Figure 20. *Median Test-/Train-loss over 60 epochs for CIFAR-100 using ResNet18, with* $\alpha = 10^{-3}$ *(Adam),* $\alpha = 10^{-4}$ *(Lion).*



(a) *Test-loss* ***(b)*** *Train-loss*

Figure 21. *Median Test-/Train-loss over 60 epochs for Tiny ImageNet using ResNet50*

E. Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.