LaProp: Separating Momentum and Adaptivity in Adam

Liu Ziyin[*][†]   Zhikang T. Wang[*]   Masahito Ueda[*][‡]

Abstract

We identity a by-far-unrecognized problem of Adam-style optimizers which results from unnecessary coupling between momentum and adaptivity. The coupling leads to instability and divergence when the momentum and adaptivity parameters are mismatched. In this work, we propose a method, LaProp, which decouples momentum and adaptivity in the Adam-style methods. We show that the decoupling leads to greater flexibility in the hyperparameters and allows for a straightforward interpolation between the signed gradient methods and the adaptive gradient methods. We experimentally show that LaProp has consistently improved speed and stability over Adam on a variety of tasks. We also bound the regret of LaProp on a convex problem and show that our bound differs from that of Adam by a key factor, which demonstrates its advantage.

1 Introduction

Modern deep learning research and application have become increasingly time-consuming due to the need to train and evaluate large models on large-scale problems, where the training can take weeks or even months to complete. This makes the study of optimization of neural networks an important field of research [1, 2]. At the core of the research lies the optimizers, i.e. the algorithms by which the parameters of neural networks are updated. Since the optimizer Adam was proposed and became widely used, various modifications of Adam have also been proposed to overcome the difficulties encountered in specific cases [3–7]. Nevertheless, none of them have shown consistent improvement over Adam on all tasks without using additional hyperparameters, and the mechanism behind remains vague.

In this paper, we propose a new adaptive optimizer, LaProp. When compared with Adam on a variety of tasks, LaProp consistently performs better or at least comparably, and especially, we find that LaProp performs better when the task is noisy or unstable, and it can optimize difficult problems for which Adam fails. Such improvement comes almost for free: LaProp is closely related to Adam, and it has exactly the same number of hyperparameters as Adam; the hyperparameter settings of Adam can be readily carried over to LaProp. Moreover, LaProp is more stable and it allows for a wider range of hyperparameter choice for which Adam would diverge, which also makes LaProp possible to reach a higher performance over it wider hyperparameter range. We

---

[*]Department of Physics and Institute for Physics of Intelligence, University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

[†]Correspondence to: Liu Ziyin ¡zliu@cat.phys.s.u-tokyo.ac.jp¿

[‡]RIKEN Center for Emergent Matter Science (CEMS), Wako, Saitama 351-0198, Japan

hope that our proposed optimizer benefits future study of deep learning and industrial applications in general.

Contribution This work has three contributions: (1) proposing a new adaptive optimization algorithm that has considerably better stability and flexibility, and confirming that such advantages indeed translate to wider applicability and better performance on tasks that are relevant to industrial applications and academic interests; (2) conceptually, extending an existing framework for understanding different optimizers, as none of the previously proposed frameworks include our method as a subclass; (3) theoretically, we give a convergence proof of our method and show that our method is different from the previous ones, i.e. ADAM and AMSGRAD, by a key factor that has limited their flexibility and may lead to worse performance.

Organization In section 2, we extend the framework in Ref. [8] to give an overview of the up-to-date optimization algorithms and introduce LAPROP as an alternative to ADAM. In section 3, we show that LAPROP is flexible and can be tuned to interpolate between the adaptive and the signed gradient methods, while it is impossible for ADAM, and in section 4, we give some mathematical properties of our method and bound its regret by $O(\sqrt{T})$. The experiments are presented in section 5.
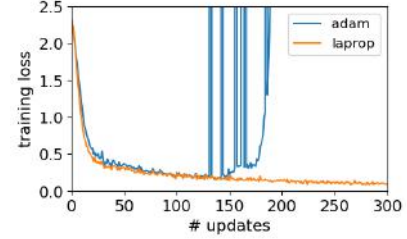

Figure 1: Divergence of ADAM on a two-layer ReLU network trained on MNIST with $\mu = 0.9$, $\nu = 0.7$. In contrast, LAPROP is always stable.

2   Conceptual Framework

Many attempts at presenting all adaptive optimizers within the same framework exist, such as in Ref. [3, 6, 9]. However, likely due to the popularization of ADAM, all those frameworks assume the structure of the ADAM algorithm, where a momentum m_t and a preconditioner n_t are computed from the gradient separately and then combined to give an update $m_t/\sqrt{n_t}$, and as a result, those frameworks do not include our adaptive optimizer as a subclass. Below, we generalize the framework in Ref. [8] to include our optimizer LAPROP, and we review the up-to-date optimization algorithms and discuss the relation between LAPROP and them under the generalized framework.

2.1   Gradient Descent

Let $\ell(\theta)$ be the loss function we want to minimize, and θ the model parameters that we optimize. To minimize ℓ, stochastic gradient descent (SGD) performs an update step $\theta_{t+1} = \theta_t - \lambda_t \nabla \ell(\theta_t)$, where λ is a step size. The optimization can be sped up by introducing momentum, which uses past update steps to accelerate the current step. The update rule can be expressed as

$$\theta_{t+1} = \theta_t - \lambda_t \nabla \ell[\theta_t - \gamma_t(\theta_t - \theta_{t-1})] + \mu_t(\theta_t - \theta_{t-1}), \qquad 0 \le \mu_t < 1, \tag{1}$$

where μ_t is the momentum hyperparameter controlling how the momentum decays. Here the factor $1 - \mu_t$ before λ_t is neglected for clarity. The Heavy-Ball momentum has $\gamma_t = 0$, and the Nesterov momentum has $\gamma_t = \mu_t$. We assume $\gamma_t = 0$ in the following discussion. Momentum has been proven to accelerate convergence [10], and it is an experimental fact that optimization of neural networks benefits from momentum [11].

2.2 The Adaptive Gradient Family

The adaptive gradient methods have emerged as the most popular tool for training deep neural networks, and they have been of great use both industrially and academically. The adaptive gradient family divides an update step by a running root mean square (RMS) of the gradients [12–14], which speeds up training by effectively rescaling the update to the order of $\lambda O(1)$. To express the adaptive gradient family, we write the generalized update rule:

$$\theta_{t+1} = \theta_t - \lambda_t G_t \nabla \ell(\theta_t) + \mu_t K_t (\theta_t - \theta_{t-1}), \tag{2}$$

where G_t and K_t are preconditioning matrices. Unlike Ref. [8], we do not assume G_t and K_t are dependent on each other. RMSPROP uses $\mu_t = 0$ and

$$G_t = \text{diag}\left((1 - \nu) \sum_{i=0}^{t} \nu^{t-i} g_t \circ g_t\right)^{-\frac{1}{2}}, \qquad g_t := \nabla \ell(\theta_t), \qquad 0 < \nu < 1, \tag{3}$$

so that the steps in SGD get divided by a RMS of the gradients, where ν is the adaptivity hyperparameter. (μ, ν are also referred to as β_1, β_2 in literature.) The most widely used method, the ADAM algorithm [14], incorporates momentum and also uses bias correction terms c_m, c_n to correct the bias in the initialization. For completeness, we first write its update rule in its most familiar form:

$$m_t = \mu m_{t-1} + (1 - \mu) \nabla \ell(\theta_t), \quad n_t = \nu n_{t-1} + (1 - \nu)(\nabla \ell(\theta_t))^2, \quad \theta_{t+1} = \theta_t - \lambda_t \frac{m_t}{c_m \sqrt{n_t / c_n}}, \tag{4}$$

where $(\nabla \ell(\theta_t))^2$ is computed element-wise, and $c_m := \frac{1}{1 - \mu^t}$ and $c_n := \frac{1}{1 - \nu^t}$, and $m_0 = n_0 = 0$. Using the framework in Equation 2, ADAM can be expressed by

$$H_t := \text{diag}\left(\frac{1 - \nu}{1 - \nu^t} \sum_{i=0}^{t} \nu^{t-i} g_t \circ g_t\right)^{\frac{1}{2}}, \quad G_t = H_t^{-1}, \quad K_t = H_t^{-1} H_{t-1}, \qquad 0 < \mu, \nu < 1, \tag{5}$$

where we have neglected factors c_m and $(1 - \mu)$ for clarity. The update rule thus becomes

$$\theta_{t+1} = \theta_t - \lambda_t H_t^{-1} \nabla \ell(\theta_t) + \mu_t \boxed{H_t^{-1} H_{t-1}} (\theta_t - \theta_{t-1}), \tag{6}$$

showing that the momentum term is reweighted by the factor in the rectangular box. The reason why ADAM has $K_t = H_t^{-1} H_{t-1}$ is because ADAM first computes an exponential average of past gradients and then divides the averaged gradient by the current preconditioner. Therefore, if we look at the parameter space, we see that the momentum of the parameter in the previous step, i.e. $\theta_t - \theta_{t-1}$, is reweighted by the new preconditioner via $H_t^{-1} H_{t-1}$, and thus the parameter-space momentum and the adaptivity are coupled. This is drastically different from the situation of SGD, where an accumulated momentum in parameter space always has additive effects and is not rescaled on future steps.

2.3 LaProp

Intuitively, the term $H_t^{-1} H_{t-1}$ in Equation 6 may have negative effects in learning. The term $H_t^{-1} H_{t-1}$ reweights the momentum by the current gradient, which can be problematic if the current gradient is pathologically large, e.g. due to noise, out-of-distribution data points or gradient

explosion, and may cause the accumulated momentum to immediately vanish and hinder learning. As an alternative to ADAM, we propose LAPROP, which simply puts $K_t = I$, i.e. the identity, and uses the following update rule:

$$\theta_{t+1} = \theta_t - \lambda_t H_t^{-1} \nabla \ell(\theta_t) + \mu_t(\theta_t - \theta_{t-1}). \tag{7}$$

As shown, LAPROP leaves the momentum of the parameter motion untouched, bringing it closer to the original momentum in non-adaptive methods. Then, even if a gradient is pathologically large, the effect of the gradient on the accumulated momentum is upper bounded by $\frac{1-\mu}{\sqrt{1-\nu}}$ by construction (see Proposition 1), which suggests that LAPROP puts more importance on its momentum than ADAM, and that a single exploding gradient is not sufficient for LAPROP to change its direction of optimization. Therefore, we conjecture that LAPROP is better at overcoming barriers, and is more stable and harder to be trapped by sharp minima. One may doubt whether such a method really converges. In section 4, we rigorously show that LAPROP indeed converges under the same assumption which ADAM requires for convergence. The LAPROP algorithm is given as Algorithm 1.

The idea of the preserved momentum in LAPROP may be related to a couple of recent works which involve parameter-space operation on neural networks [15–18]. In those works, the parameters are directly averaged or decayed, and faster, more stable and more generalizable results are obtained. While the momentum mechanism in ADAM is designed to average the gradient, LAPROP averages the parameter updates throughout the training process, which is in agreement with the recent discovery of the effectiveness of parameter-space averaging.

Algorithm 1 LAPROP

Input: $\theta_1 \in \mathbb{R}^d$, learning rate $\{\lambda_t\}_{t=1}^T$, decay parameters $0 \leq \mu, \nu < 1$, and $\epsilon \ll 1$, bias correction factors $0 < c_n, c_m < 1$. Set $m_0 = n_0 = 0$.
for $t = 1$ **to** T **do**
  $g_t = \nabla_\theta \ell(\theta_t)$
  $n_t = \nu n_{t-1} + (1 - \nu)g_t^2$
  $m_t = \mu m_{t-1} + (1 - \mu)\frac{g_t}{\sqrt{n_t/c_n} + \epsilon}$
  $\theta_{t+1} = \theta_t - \lambda_t m_t/c_m$
end for

---

3   Interpolation Between Adaptive and Signed Gradient Methods

An immediate consequence of LAPROP is that we can tune $\nu \to 0$ to recover the signed stochastic gradient (SSG) methods [19], where the magnitude of the gradient is ignored and only the sign is used for optimization. Specifically, for $\nu = 0$, we have $n_t = g_t^2$ and $m_t = \mu m_{t-1} + (1 - \mu)\frac{g_t}{|g_t|}$ in Algorithm 1, and the LAPROP update rule becomes $\theta_{t+1} = \theta_t - (1 - \mu)\lambda_t \mathrm{sgn}[\nabla \ell(\theta_t)] + \mu(\theta_t - \theta_{t-1})$. However, the same is generally not true for ADAM.

3.1   Divergence of Adam

If we set $\nu = 0$ and $\mu \neq 0$ for ADAM, ADAM will diverge. To see how it happens, we assume that the gradients g_t are i.i.d. drawn from a normal distribution, and for ADAM we have

$$\mathrm{Var}\left[\lim_{\nu \to 0} \frac{m_t}{\sqrt{n_t}}\right] = \mathrm{Var}\left[\frac{\sum_i^t \mu^{t-i} g_i}{g_t}\right] \geq \mathrm{Var}\left[\mu \frac{g_{t-1}}{g_t}\right] = \infty \qquad \text{for } t > 1, \tag{8}$$

and thus one cannot divide the accumulated gradient by the current preconditioner as done by ADAM. Nevertheless, ADAM can still recover SSG for the special case of $\mu = \nu = 0$. The behaviors of ADAM and LAPROP for the limiting values of μ and ν are summarized in Table 1.

Table 1: The relationship of ADAM and LAPROP to other methods, where SSG-M represents SSG with momentum, and we refer to the bias-corrected version of RMSPROP. In all the limiting cases of μ and ν, LAPROP asymptotes to reasonable algorithms, while ADAM does not for $\nu \to 0$ and $\mu > 0$. This table also shows that the signed gradient family are special cases of LAPROP. We emphasize that although ADAM and LAPROP become asymptotically equivalent at the limit of $\nu \to 1$, they still show different behaviours even at $\nu = 0.999$.

	param.	$\nu \to 0$	$\nu \to 1$
ADAM	$\mu \to 0$	SSG	RMSPROP
	$\mu \to 1$	*divergence*	ADAM(μ, ν)
LAPROP	$\mu \to 0$	SSG	RMSPROP
	$\mu \to 1$	SSG-M	ADAM(μ, ν)

The above suggests that μ and ν have complex relation in ADAM. As μ controls the averaging of the past gradients on the numerator and ν controls the averaging on the denominator, if μ and ν do not match, ADAM may diverge. This issue also appears in the convergence analysis of ADAM [3, 20]. For $\sqrt{\nu} > \mu$, we can prove that an update step of ADAM is upper bounded by $\frac{1}{1 - \mu/\sqrt{\nu}}$ (Proposition 2), but for $\sqrt{\nu} < \mu$, we are not aware of a bound. In contrast, the update of LAPROP is always upper bounded by $\frac{1}{\sqrt{1-\nu}}$ (Proposition 1). The same is demonstrated in Figure 1, where ADAM diverges on a simple task if ν and μ are not set appropriately. Similarly, we expect that ADAM becomes unstable with a decreasing ν, especially if the past gradients are noisy, or if the past gradients do not come from the same distribution as the learning proceeds.

3.2 Advantage of Interpolating to Signed Stochastic Gradient Methods

SSG uses the simple update rule $\theta_{t+1} = \theta_t - (1 - \mu)\lambda_t \text{sgn}[\nabla \ell(\theta_t)] + \mu(\theta_t - \theta_{t-1})$.[1] This simple method is surprisingly efficient in training neural networks and it can be comparable to SGD and the adaptive methods, as shown in Ref. [19]. Especially, it is found to be effective when the signal-to-noise ratio is low, which is also evidenced by our experiments in section 5.1. Furthermore, Ref. [21] suggests that the sign of the gradient actually accounts for most of the improvement offered by the adaptive gradient methods, and that the adaptive gradient methods may be interpreted as using the variance of the gradient to adaptively rescale the step in SSG. Therefore, it is reasonable to consider an interpolation from the adaptive methods to the SSG methods, i.e. to gradually tune off the adaptivity. As we have shown, only LAPROP can realize such an interpolation in general.

The interpolation to SSG offers two advantages for LAPROP. The first advantage is that with the presence of the interpolation, LAPROP always has stable and reasonable behaviour for all different hyperparameter settings. In our various experiments, we notice that if fine-tuning is needed, the fine-tuning of LAPROP is easier than that of ADAM, and we see that LAPROP changes stably and smoothly regarding its varying hyperparameters, as in section 5.1 and 5.2. The easier fine-tuning may save a great amount of effort if a large-scale task is involved. The second advantage is that since SSG is effective when the noise is large, LAPROP may be tuned to the SSG side according to the noise level of the task and achieve higher performance or faster speed. For some tasks where the training is unstable only at the beginning, it is possible to use a small ν at the beginning and

---

[1]Ref. [19] proposed a slightly different version which averages the gradient as the momentum and then takes its sign. We experimentally find that our version is qualitatively the same as theirs.

change to $\nu \to 1$ later to obtain faster final convergence.

In summary, LaProp decouples the parameter-space momentum from the adaptivity so that the effects of μ and ν are independent of each other, which leads to more flexibility, stability and better performance, and gives the optimizer more tunable behaviour than Adam. The default LaProp hyperparameters we recommend are $\lambda = 4 \times 10^{-4}$, $\mu = 0.8$ - 0.9, $\nu = 0.95$ - 0.999 and $\epsilon = 10^{-15}$. Nevertheless, hyperparameter settings of Adam always work for LaProp. If the task is noisy or complex, one may try $\nu = 0$ - 0.9 in the beginning and probably increase ν to a larger value later. LaProp is found to be highly stable and it works across different hyperparameter settings, and it is much less sensitive to its hyperparameter settings when compared with Adam.

4   Mathematical Properties and Convergence Analysis

The most important property of LaProp is probably that its update is always bounded:

Proposition 1. *Bound for* LaProp *update. Let m_t be defined as in Algorithm 1, and set $c_n = 1 - \nu^t$, $c_m = 1 - \mu^t$. Then the magnitude of the update is bounded from above as $\left| \frac{m_t}{c_m} \right| \leq \frac{1}{\sqrt{1-\nu}}$.*

An important feature of this bound is that it only depends on ν. This is in contrast with the analysis for Adam:

Proposition 2. *Bound for Adam update. Let m_t, n_t, c_n, c_m be defined as in Equation 4, and set $c_n = 1 - \nu^t$, $c_m = 1 - \mu^t$. Assume $\mu < \sqrt{\nu}$. Then the magnitude of the update is bounded from above as $\left| \frac{m_t}{c_m \sqrt{n_t/c_n}} \right| \leq \frac{1}{1-\gamma}$, where $\gamma = \frac{\mu}{\sqrt{\nu}}$.*

The proofs are given in the appendix. Notice that there are two key points: (1) the bound depends on the the ratio $\frac{\mu}{\sqrt{\nu}}$, suggesting that the momentum μ and the adaptivity ν are coupled; (2) the bound only applies when $\mu > \sqrt{\nu}$, suggesting that the range of μ and ν is limited. Although the default setting of Adam is ($\mu = 0.9$, $\nu = 0.999$) which is within the bound, yet as our experiments have demonstrated, with such a restriction removed, LaProp can potentially achieve much more.

Next, we present the convergence theorem of LaProp in a convex setting. The proof closely follows the results of the Adam style optimizers in Ref. [3, 14], and most of the derivations thereof can be readily applied to LaProp. Note that a rigorous proof of convergence for the adaptive gradient family has been a major challenge, with various bounds and rates present. In this section, we aim to present a bound whose terms are qualitatively insightful when compared with those of Adam.

Theorem 1. *(Regret bound for convex problem) Let the loss function be convex and the gradient be bounded, with $\|\nabla \ell_t(\theta)\|_\infty \leq G_\infty$ for all $\theta \in \mathbb{R}^d$, and let the distance between any θ_t learned by* LaProp *be bounded, with $\|\theta_{t_1} - \theta_{t_2}\|_\infty \leq D$, and let $\mu, \nu \in [0, 1)$. Let the learning rate and the momentum decay as $\lambda_t = \lambda/\sqrt{t}$, $\mu_t = \mu \zeta^{t-1}$ for $\zeta \in (0, 1)$. Define $s_t := \sqrt{\frac{n_t}{c_n}}$. If we assume $\frac{s_{t+1}}{\lambda_{t+1}} \geq \frac{s_t}{\lambda_t}$, then the regret $R(T) := \sum_t^T (\ell(\theta_t) - \ell(\theta^*))$ can be bounded from above as*

$$R(T) \leq \frac{D^2 \sqrt{T}}{2\lambda(1-\mu)} \sum_{i=1}^d s_{T,i} + \frac{\mu d D^2 G_\infty}{2\lambda(1-\mu)(1-\zeta)^2} + \frac{\lambda\sqrt{1 + \ln T}}{(1-\mu)(1-\nu)\sqrt{1-\nu}} \sum_{i=1}^d \|g_{1:T,i}\|_2, \qquad (9)$$

where in the worst case, we have $R(T) \leq \frac{D^2\sqrt{T}}{2\lambda(1-\mu)} \sum_{i=1}^d s_{T,i} + \frac{\mu d D^2 G_\infty}{2\lambda(1-\mu)(1-\zeta)^2} + \frac{2\lambda d G_\infty \sqrt{T}}{(1-\mu)(1-\nu)}$.
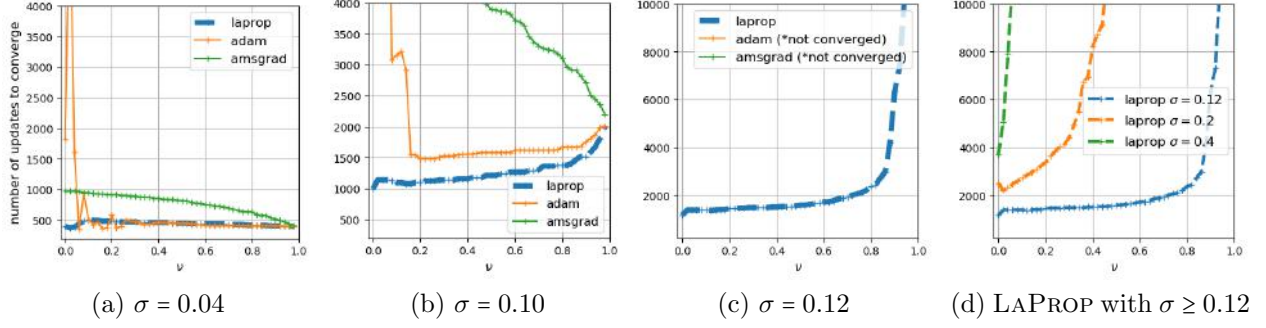
Figure 2: Time it takes to converge on the noisy Rosenbrock task plotted against ν, with σ being the noise level. (a) When the noise is small, the optimization speed of LAPROP is almost invariant w.r.t. ν, demonstrating its flexibility in hyperparameters compared with the other optimizers; (b) when the noise gets larger, the performance of ADAM and AMSGRAD decreases, and they cannot work in the small ν regime where LAPROP has its best performance; (c, d) For $\sigma \geq 0.12$, only LAPROP converges even if we lengthen the optimization to 10000 steps. Data points are plotted at equal intervals for all the curves, and we see LAPROP is much stabler.

We see that the major difference between this bound and that of AMSGRAD in Ref. [3] lies in the third term, where LAPROP replaces the factor $\frac{1}{(1-\mu)(1-\gamma)}$ by $\frac{1}{1-\nu}$.[2] LAPROP converges for any $\nu \in [0,1)$, while the variants of ADAM depends on the relation between μ and ν, i.e. $\sqrt{\nu} > \mu$ must hold true. As a side note, the assumption $\frac{s_{t+1}}{\lambda_{t+1}} - \frac{s_t}{\lambda_t} \geq 0$ is crucial, which also appears in the proof for the convergence of ADAM and other known adaptive optimizers [9, 14]. This assumption is shown to be necessary for the convergence of ADAM [3], and we find that LAPROP also converges provided with such an assumption.

From the above theorem, one see that the average regret of LAPROP converges at the same asymptotic rate as SGD and other adaptive optimizers such as ADAM and AMSGRAD, i.e. $\frac{R(T)}{T} \sim O\left(\frac{1}{\sqrt{T}}\right)$. Like ADAM and AMSGRAD, the above regret bound can be considerably better than $O(\sqrt{dT})$ if the gradient is sparse, as shown in Ref. [12]. The experimental fact that the adaptive gradient family is faster at training neural networks suggests that the gradient is often indeed sparse.

5   Experiment

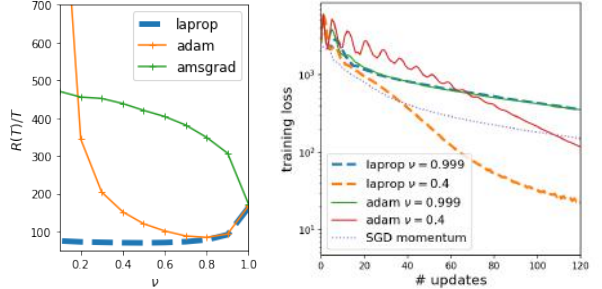### 5.1   Optimizing the Noisy Rosenbrock Loss

In this section, we experimentally show the improved performance of LAPROP compared with the previous methods, especially ADAM. Detailed settings are given in the appendix. First, we consider optimization of the Rosenbrock loss function [22], for which the optimum is known exactly. Given two parameters (x, y), the noisy Rosenbrock loss is defined as $\ell(x, y) = (1 - (x + \epsilon_1))^2 + 100\left((y + \epsilon_2) - (x + \epsilon_1)^2\right)^2$, where ϵ_1 and ϵ_2 are noise terms. At each update step, ϵ_1 and ϵ_2 are independently sampled from $Uniform(-\sigma, \sigma)$, and thus the loss landscape is effectively shifted by ϵ_1 and ϵ_2. Parameters x, y are initialized to be 0, and the optimal solution is $(1, 1)$. When (x, y) is sufficiently optimized such that $(1 - x)^2 + 100\left(y - x^2\right)^2 < 0.1$ holds, we say that the optimization is successful and it has converged. As shown in Figure 2, ADAM, AMSGRAD and LAPROP have

---

[2]Compared with Ref. [3], the second term in the bound is refined and it does not include the square of $\frac{1}{1-\mu}$.

different behaviors on this simple task. When the noise is small, LaProp is one of the fastest method, but when the noise gets large, the task becomes increasingly difficult and only LaProp can work. It is worth noting that LaProp is more stable and it is less sensitive to the hyperparameter ν. Its performance changes with ν smoothly. Hyperparameters of the three optimizers are always identical in the experiment.

5.2 Neural Style Transfer

Following the optimization task above, we consider the task of generating an image with a given content and a given style, which is also purely optimizational and it is described in detail in Ref. [23]. As shown in Figure 3(a), we still see LaProp performs best across different ν values. Although the trend in Figure 3(a) agrees well with that in section 5.1, actually, the best ν value for LaProp is non-zero and is found to be 0.4, suggesting that the tuning of ν is nontrivial and may indeed improve the performance. On this task also, lower ν offers better performance for LaProp and worse performance for AMS-

(a) regret vs. ν (b) optimization trajectories

Figure 3: Neural Style Transfer with different optimizers. (a) The average regret $R(T)/T$ at $T = 1000$ plotted against ν. A lower value corresponds to a better convergence rate [3, 14]. (b) Example optimization curves of different optimizers for the first 120 updates.

Grad, and Adam performs well when ν is relatively large. Examples of the optimization curves are shown in Figure 3(b). We see that while LaProp is stable and achieves its fastest speed at $\nu = 0.4$, Adam exhibits oscillatory behavior at the same ν.
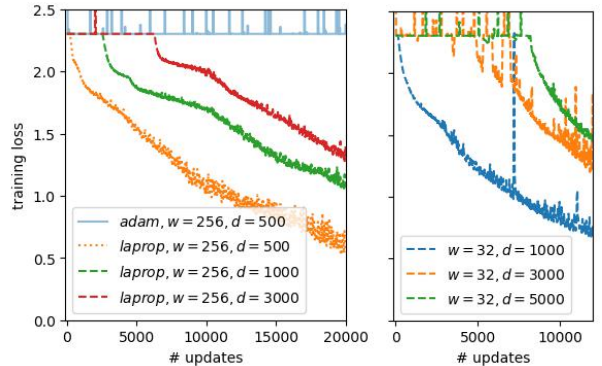
5.3 Training Extremely Deep Fully Connected Networks

In the following, we demonstrate that LaProp can be used as a reliable substitute for Adam concerning deep learning, and it perform comparably to or outperforms Adam on a variety of tasks even for $\nu \sim 1$, which is the default of Adam. First, we show that LaProp has stronger ability to deal with hard optimization problems in deep learning. Specifically, we show that LaProp can train extremely deep fully connected (FC) networks on MNIST with the ReLU activation. Ref. [24] shows that it is hard to initialize the training of FC networks at extreme depth due to the gradient explosion or vanishing problem, where SGD, Adam, and AMSGrad all fail. Nevertheless, we surprisingly find that LaProp can train neural networks

(a) LaProp and Adam   (b) LaProp only

Figure 4: Training curves of deep FC networks. Only LaProp can train when there are 500 layers or more.

with a depth up to 5000 (i.e. already the deepest that our GPU can afford). The results are shown in Figure 4, where we consider networks that are composed of d fully connected layers all of which have w neurons, using the Kaiming uniform initialization. Here we have used $\nu = 0.96$. To our best try, none of Adam, AMSGrad or SGD can train FC networks of such depths, which is in

8

(a) training of IWSLT14 de-en (b) training RoBERTa on the full English Wikipedia
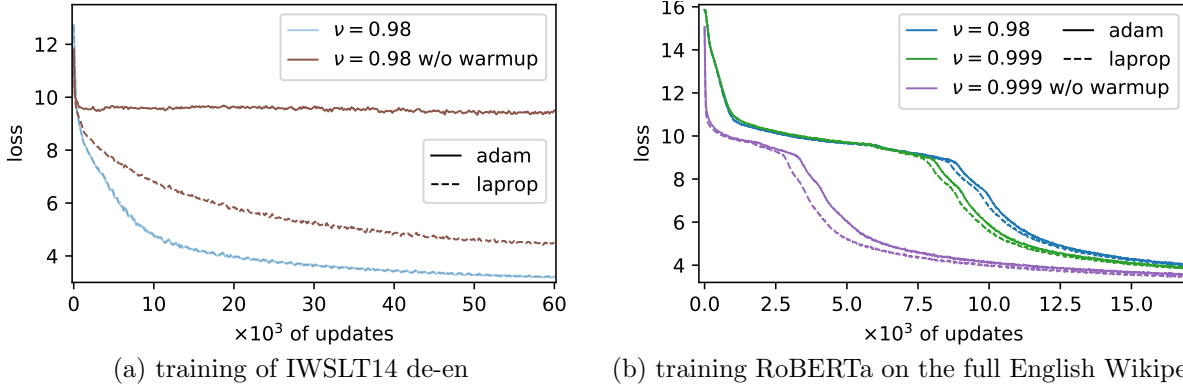
Figure 5: Learning curves of the transformer tasks. When there is a warmup, the learning rate linearly increases from zero to the maximum and then decreases; otherwise it starts from the maximum and decreases. The warmup includes the first 2×10^3 updates in (a), and 10×10^3 updates in (b).

agreement with Ref. [24]. For illustration, we plot ADAM for $d = 500$. These results indicate that LAPROP is considerably more robust and powerful for handling complex and pathological training problems.

5.4 Translation and Pretraining with Transformers

As a recently proposed modern network architecture [25], the transformer is known to be difficult to train. Especially, it usually requires a warm-up schedule of learning rates if a large learning rate is to be used, and it is typically trained by ADAM with $\nu \leq 0.999$. We show that the performance and the optimization speed of LAPROP parallel or outperform those of ADAM. We consider the tasks of IWSLT14 German to English translation [26] and the large-scale RoBERTa(base) pretraining using the full English Wikipedia [27], which are of both industrial and academic interest. We follow the prescriptions given by Ref. [28] and use the default network architectures and settings thereof, and the results are shown in Figure 5. For the IWSLT14 translation task, when a warmup is used, the learning curves of ADAM and LAPROP coincide; without a warmup, as is also shown in Ref. [4], ADAM gets trapped by a bad minimum and does not converge to a low loss, while we find that LAPROP does not get completely trapped and it continues looking for a better solution, indicating its higher capability of overcoming barriers. For the RoBERTa pretraining, we find that in our setting, we can use $\nu = 0.999$ as well as the default $\nu = 0.98$, and we can safely ignore the warmup. We find that in all the cases, the speed of LAPROP is faster than ADAM, and if the optimization speed of the problem becomes faster, LAPROP outperforms ADAM even more. We also notice that a smaller ν speeds up the training at the initial stage while a larger ν speeds up later, which implies that a ν schedule may be beneficial. Details are given in the appendix.

5.5 Reinforcement Learning of Atari Games

We compare the performance of LAPROP and ADAM on a modern reinforcement learning (RL) task, i.e. learning to play Atari2600 games, and we use the Rainbow DQN algorithm in Ref. [29]. As reported in the original paper Ref. [29], the full training of all the games takes a month, which shows the difficulty of the task. Here, a few minor settings of the default Rainbow DQN have been modified by us to accelerate training. Since the task is difficult and slow, we only investigate
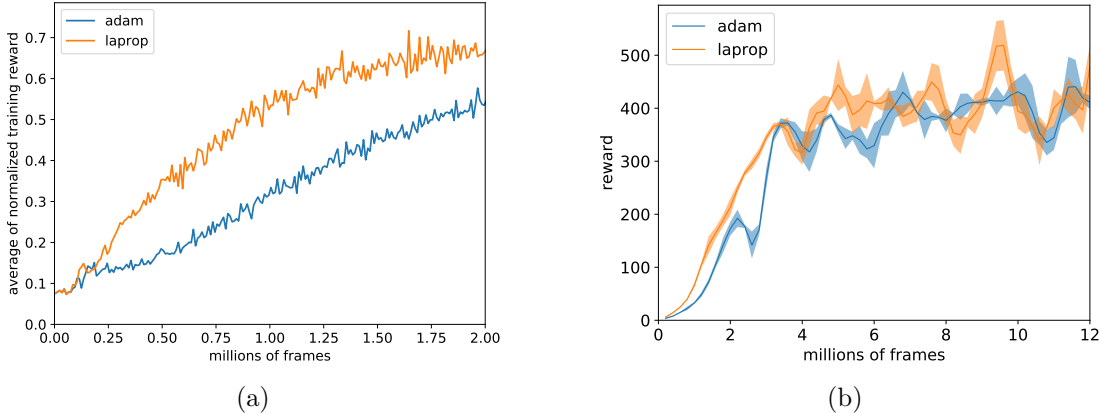
Figure 6: (a) Average of the normalized training performance on 20 different Atari2600 games, for which the learning shows clear progress at an early stage. (b) Test performance on Atari2600 game *Breakout*. After every 0.2 million frames the trained model is evaluated for 20 times to obtain a test performance, which is plotted with one standard error. Gaussian smoothing with a standard deviation of 1 is applied to the nearby test performance data points.

the final performance of training on a single game, and we compare the early-stage performance of LAPROP and ADAM on other games. The results are shown in Figure 6. The training performances of LAPROP and ADAM are averaged on 20 different Atari2600 games after being normalized, and are shown in Figure 6(a). The evaluated test performance on the Atari2600 game *Breakout* is shown in Figure 6(b). Details and the results of all the 20 games are given in the appendix. We find that LAPROP always starts learning earlier than ADAM and often achieves high performances faster than ADAM. Its performance improves with fluctuation just like ADAM, but its overall performance is better. It is important to realize that the fluctuation of performance is a consequence of RL and is not only due to noise. For the game *Breakout*, throughout the training, the best trained model obtained by LAPROP has a test performance of 607.3 ± 47.6, which is higher than that of ADAM, i.e. 522.9 ± 40.6. Therefore we see that LAPROP can be used to achieve both faster and better RL results. In our experiments, we also notice that LAPROP is qualitatively more aggressive than ADAM in learning and exploration.

6   Conclusion

Based on and motivated by a series of previous works, we have proposed an optimization method that is shown to be effective for training modern neural networks on various tasks. While the proposed method does outperform and show better flexibility than other members in the adaptive gradient family, we remark that the understanding of its true advantage needs to be tested with more experiments and be investigated in further detail. Additional experimental results on CIFAR10 using residual networks and on IMDB using LSTM are provided in the appendix.

10

References

[1] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[2] Ruoyu Sun. Optimization for deep learning: theory and algorithms. *arXiv preprint arXiv:1912.08957*, 2019.

[3] Sashank Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018.

[4] Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. *arXiv preprint arXiv:1908.03265*, 2019.

[5] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[6] Liangchen Luo, Yuanhao Xiong, Yan Liu, and Xu Sun. Adaptive gradient methods with dynamic bound of learning rate. In *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, Louisiana, May 2019.

[7] Jinghui Chen, Dongruo Zhou, Yiqi Tang, Ziyan Yang, and Quanquan Gu. Closing the generalization gap of adaptive gradient methods in training deep neural networks. *arXiv preprint arXiv:1806.06763*, 2018.

[8] Ashia C Wilson, Rebecca Roelofs, Mitchell Stern, Nati Srebro, and Benjamin Recht. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pages 4148–4158, 2017.

[9] Xiangyi Chen, Sijia Liu, Ruoyu Sun, and Mingyi Hong. On the convergence of a class of adam-type algorithms for non-convex optimization. *arXiv preprint arXiv:1808.02941*, 2018.

[10] Yurii Evgen'evich Nesterov. A method of solving a convex programming problem with convergence rate $o(k^2)$. In *Doklady Akademii Nauk*, volume 269, pages 543–547. Russian Academy of Sciences, 1983.

[11] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013.

[12] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011.

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

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[15] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[16] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[17] Michael Zhang, James Lucas, Jimmy Ba, and Geoffrey E Hinton. Lookahead optimizer: k steps forward, 1 step back. In *Advances in Neural Information Processing Systems*, pages 9593–9604, 2019.

[18] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry P Vetrov, and Andrew G Wilson. Loss surfaces, mode connectivity, and fast ensembling of dnns. In *Advances in Neural Information Processing Systems*, pages 8789–8798, 2018.

[19] Jeremy Bernstein, Yu-Xiang Wang, Kamyar Azizzadenesheli, and Anima Anandkumar. signsgd: Compressed optimisation for non-convex problems. *arXiv preprint arXiv:1802.04434*, 2018.

[20] André Belotto da Silva and Maxime Gazeau. A general system of differential equations to model first order adaptive algorithms. *arXiv preprint arXiv:1810.13108*, 2018.

[21] Lukas Balles and Philipp Hennig. Dissecting adam: The sign, magnitude and variance of stochastic gradients. *arXiv preprint arXiv:1705.07774*, 2017.

[22] HoHo Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

[23] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[24] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. In *Advances in Neural Information Processing Systems*, pages 571–581, 2018.

[25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[26] Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign, iwslt 2014. In *Proceedings of the International Workshop on Spoken Language Translation, Hanoi, Vietnam*, page 57, 2014.

[27] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[28] Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. fairseq: A fast, extensible toolkit for sequence modeling. In *Proceedings of NAACL-HLT 2019: Demonstrations*, 2019.

[29] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[31] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[32] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[33] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[34] Jian Guo, He He, Tong He, Leonard Lausen, Mu Li, Haibin Lin, Xingjian Shi, Chenguang Wang, Junyuan Xie, Sheng Zha, Aston Zhang, Hang Zhang, Zhi Zhang, Zhongyue Zhang, and Shuai Zheng. Gluoncv and gluonnlp: Deep learning in computer vision and natural language processing. *arXiv preprint arXiv:1907.04433*, 2019.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[36] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

A Mathematical Proofs

A.1 Proof of Proposition 1

Expand the update step:

$$
\begin{aligned}
|m_t| &\leq \mu|m_{t-1}| + (1-\mu)\frac{|g_t|}{\sqrt{n_t/c_n(t)}} \\
&\leq \mu|m_{t-1}| + (1-\mu)\frac{|g_t|}{\sqrt{[(1-\nu)g_t^2 + \nu n_{t-1}]/c_n(t)}} \\
&\leq \mu|m_{t-1}| + (1-\mu)\frac{|g_t|}{\sqrt{[(1-\nu)g_t^2]/c_n(t)}} \\
&\leq \mu|m_{t-1}| + \frac{1-\mu}{\sqrt{1-\nu}}
\end{aligned}
\tag{10}
$$

and this defines a recurrence relation that solves to

$$
|m_t| \leq \frac{1-\mu^t}{\sqrt{1-\nu}}
\tag{11}
$$

whereby

$$
\frac{|m_t|}{c_m(t)} \leq \frac{1}{\sqrt{1-\nu}}.
\tag{12}
$$

A.2 Proof of Proposition 2

Expand the term:

$$
\begin{aligned}
\frac{|m_t|}{\sqrt{n_t}} &\leq \mu\frac{|m_{t-1}|}{\sqrt{n_t}} + (1-\mu)\frac{|g_t|}{\sqrt{n_t}} \\
&\leq \mu\frac{|m_{t-1}|}{\sqrt{\nu n_{t-1}}} + (1-\mu)\frac{|g_t|}{\sqrt{(1-\nu)g_t^2}} \\
&\leq \frac{1-\mu}{\sqrt{1-\nu}}\sum_{t'=0}^{t}\frac{\mu^{t'}}{\nu^{\frac{t'}{2}}}
\end{aligned}
\tag{13}
$$

and if $\mu < \sqrt{\nu}$, then

$$
\frac{|m_t|}{\sqrt{n_t}} \leq \frac{1-\mu}{\sqrt{1-\nu}}\frac{1}{1-\gamma}
\tag{14}
$$

where $\gamma = \frac{\mu}{\sqrt{\nu}}$. Putting in the bias corrections, we obtain that

$$
\frac{|m_t|/c_m(t)}{\sqrt{n_t/c_n(t)}} \leq \frac{1}{\sqrt{1-\nu}}\frac{1}{1-\gamma},
\tag{15}
$$

and we are done.

A.3 Proof of Theorem 1

By convexity, we have

$$\ell(\theta_t) - \ell(\theta^*) \leq \sum_{i=1}^{d} g_{t,i}(\theta_{t,i} - \theta_i^*), \tag{16}$$

where θ^* is the global minimum of ℓ. In the following, we focus on a single component with index i and bound the term $g_{t,i}(\theta_{t,i} - \theta_i^*)$. We plug in the LAPROP update rule and obtain

$$
\begin{aligned}
\theta_{t+1} &= \theta_t - \lambda_t \frac{m_t}{c_m} \\
&= \theta_t - \frac{\lambda_t}{c_m}\left(\mu_t m_{t-1} + (1 - \mu_t)\frac{g_t}{s_t}\right)
\end{aligned}
\tag{17}
$$

where we have defined $s_t := \sqrt{n_t/c_n}$. We subtract θ^* from both sides and square them to obtain

$$(\theta_{t+1} - \theta^*)^2 = (\theta_t - \theta^*)^2 - 2\frac{\lambda_t}{c_m}\left(\mu_t m_{t-1} + (1 - \mu_t)\frac{g_t}{s_t}\right)(\theta_t - \theta^*) + \frac{\lambda_t^2}{c_m^2}m_t^2. \tag{18}$$

The terms are rearranged to give obtain an expression for $g_t(\theta_t - \theta^*)$ using the second term above. In the second line below, we use the inequality $ab < \frac{a^2+b^2}{2}$.

$$
\begin{aligned}
g_t(\theta_t - \theta^*) &= \frac{c_m}{2\lambda_t(1 - \mu_t)}\left[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2\right]s_t + \frac{\mu_t}{1 - \mu_t}s_t m_{t-1}(\theta^* - \theta_t) + \frac{\lambda_t}{2c_m(1 - \mu_t)}s_t m_t^2 \\
&\leq \frac{c_m}{2\lambda_t(1 - \mu_t)}\left[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2\right]s_t + \frac{\lambda_t}{2c_m(1 - \mu_t)}s_t m_t^2 \\
&\qquad + \frac{\mu_t}{2\lambda_t(1 - \mu_t)}(\theta^* - \theta_t)^2 s_t + \frac{\lambda_t \mu_t}{2(1 - \mu_t)}s_t m_{t-1}^2
\end{aligned}
\tag{19}
$$

Proposition 1 is applied to bound m_t.

$$
\begin{aligned}
g_t(\theta_t - \theta^*) &\leq \frac{c_m}{2\lambda_t(1 - \mu_t)}\left[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2\right]s_t + \frac{\lambda_t}{2(1 - \mu_t)(1 - \nu)}s_t \\
&\qquad + \frac{\mu_t}{2\lambda_t(1 - \mu_t)}(\theta^* - \theta_t)^2 s_t + \frac{\lambda_t \mu_t}{2(1 - \mu_t)(1 - \nu)}s_t \\
&\leq \frac{c_m}{2\lambda_t(1 - \mu_t)}\left[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2\right]s_t + \frac{\mu_t}{2\lambda_t(1 - \mu)}(\theta^* - \theta_t)^2 s_t + \frac{\lambda_t}{(1 - \mu)(1 - \nu)}s_t
\end{aligned}
\tag{20}
$$

Now we are ready to bound $R(T)$, to do this, we sum over the index i and the times steps from 1 to T (recall that each of m, θ, s carries an index i):

$$R(T) \leq \sum_{i=1}^{d} \sum_{t=1}^{T} \left\{ \frac{c_{m,t}}{2\lambda_t(1-\mu_t)} \left[(\theta_t - \theta^*)^2 - (\theta_{t+1} - \theta^*)^2 \right] s_t + \frac{\mu_t}{2\lambda_t(1-\mu)} (\theta^* - \theta_t)^2 s_t + \frac{\lambda_t}{(1-\mu)(1-\nu)} s_t \right\}$$

$$= \sum_{i=1}^{d} \frac{c_{m,1}}{2\lambda_1(1-\mu_1)} (\theta_{1,i} - \theta_i^*)^2 s_{1,i} + \sum_{i=1}^{d} \sum_{t=2}^{T} \frac{1}{2} (\theta_{t,i} - \theta_i^*)^2 \underbrace{\left(\frac{c_{m,t} s_{t,i}}{(1-\mu_t)\lambda_t} - \frac{c_{m,t-1} s_{t-1,i}}{(1-\mu_{t-1})\lambda_{t-1}} \right)}_{A}$$

$$+ \sum_{i=1}^{d} \sum_{t=1}^{T} \frac{\mu_t s_{t,i}}{2\lambda_t(1-\mu)} (\theta_i^* - \theta_{t,i})^2 + \sum_{i=1}^{d} \sum_{t=1}^{T} \frac{\lambda_t}{(1-\mu)(1-\nu)} s_{t,i}.$$

$$(21)$$

We focus on the term A. Using $\mu_t < \mu_{t-1}$, we have

$$A \leq \frac{c_{m,t} s_{t,i}}{(1-\mu_{t-1})\lambda_t} - \frac{c_{m,t-1} s_{t,i}}{(1-\mu_{t-1})\lambda_t} = \frac{1}{1-\mu_{t-1}} \left(\frac{c_{m,t} s_{t,i}}{\lambda_t} - \frac{c_{m,t-1} s_{t-1,i}}{\lambda_{t-1}} \right). \tag{22}$$

Also with $c_{m,t-1} \leq c_{m,t} \leq 1$ and the assumption $\frac{s_{t-1,i}}{\lambda_{t-1}} \leq \frac{s_{t,i}}{\lambda_t}$, we have

$$\frac{c_{m,t} s_{t,i}}{\lambda_t} - \frac{c_{m,t-1} s_{t-1,i}}{\lambda_{t-1}} \geq 0. \tag{23}$$

Therefore we obtain

$$R(T) \leq \sum_{i=1}^{d} \frac{c_{m,1}}{2\lambda_1(1-\mu_1)} (\theta_{1,i} - \theta_i^*)^2 s_{1,i} + \sum_{i=1}^{d} \sum_{t=2}^{T} \frac{1}{2} (\theta_{t,i} - \theta_i^*)^2 \left(\frac{c_{m,t} s_{t,i}}{(1-\mu_{t-1})\lambda_t} - \frac{c_{m,t-1} s_{t-1,i}}{(1-\mu_{t-1})\lambda_{t-1}} \right)$$

$$+ \sum_{i=1}^{d} \sum_{t=1}^{T} \frac{\mu_t s_{t,i}}{2\lambda_t(1-\mu)} (\theta_i^* - \theta_{t,i})^2 + \sum_{i=1}^{d} \sum_{t=1}^{T} \frac{\lambda_t}{(1-\mu)(1-\nu)} s_{t,i}$$

$$\leq \sum_{i=1}^{d} \frac{c_{m,1} s_{1,i}}{2\lambda_1(1-\mu)} D^2 + \sum_{i=1}^{d} \sum_{t=2}^{T} \frac{D^2}{2(1-\mu)} \left(\frac{c_{m,t} s_{t,i}}{\lambda_t} - \frac{c_{m,t-1} s_{t-1,i}}{\lambda_{t-1}} \right) + \sum_{i=1}^{d} \sum_{t=1}^{T} \frac{\mu_t s_{t,i}}{2\lambda_t(1-\mu)} D^2 \tag{24}$$

$$+ \sum_{i=1}^{d} \sum_{t=1}^{T} \frac{\lambda_t}{(1-\mu)(1-\nu)} s_{t,i}$$

$$\leq \frac{D^2 \sqrt{T}}{2\lambda(1-\mu)} \sum_{i=1}^{d} s_{T,i} + \frac{\mu d D^2 G_\infty}{2\lambda(1-\mu)(1-\zeta)^2} + \frac{\lambda}{(1-\mu)(1-\nu)} \underbrace{\sum_{i=1}^{d} \sum_{t=1}^{T} \frac{s_{t,i}}{\sqrt{t}}}_{B}.$$

Note that we have used $\|\theta_{t_1} - \theta_{t_2}\|_\infty \leq D$ and $s_{t,i} \leq G_\infty$, and the second last term is derived using $\frac{\mu_t}{\lambda_t} = \frac{\mu \zeta^t \sqrt{t}}{\lambda} \leq \frac{\mu t \zeta^t}{\lambda}$. We now try to bound the term B above. The worst case bound can be found by using $s_{t,i} \leq G_\infty$ and $\sum_t^T 1/\sqrt{t} < 2\sqrt{T} - 1$. Thus the worst case bound is $B \leq 2 d G_\infty \sqrt{T}$, which is the same as other adaptive gradient methods.

Another bound, which is tighter when the gradient is sparse, can be obtained using the Cauchy-

Schwarz inequality as follows:

$$
\begin{aligned}
\sum_{i=1}^{d}\sum_{t=1}^{T}\frac{s_{t,i}}{\sqrt{t}} &\leq \sum_{i=1}^{d}\sqrt{\sum_{t}\frac{n_{t,i}}{c_n}}\sqrt{\sum_{t}\frac{1}{t}} \\
&\leq \sqrt{1+\ln T}\sum_{i=1}^{d}\sqrt{\sum_{t}\frac{n_{t,i}}{c_n}} \\
&= \sqrt{1+\ln T}\sum_{i=1}^{d}\sqrt{\sum_{t}^{T}\frac{\sum_{j}^{t}(1-\nu)\nu^{t-j}g_{j,i}^2}{c_n}} \\
&= \sqrt{1+\ln T}\sum_{i=1}^{d}\sqrt{\sum_{t}^{T}\sum_{j}^{t}\nu^{t-j}g_{j,i}^2} \\
&\leq \sqrt{\frac{1+\ln T}{1-\nu}}\sum_{i=1}^{d}\|g_{1:T,i}\|
\end{aligned}
\tag{25}
$$

and so we obtain the desired bound

$$
R(T) \leq \frac{D^2\sqrt{T}}{2\lambda(1-\mu)}\sum_{i=0}^{d}s_{T,i} + \frac{\mu dD^2 G_\infty}{2\lambda(1-\mu)(1-\zeta)^2} + \frac{\lambda\sqrt{1+\ln T}}{(1-\mu)(1-\nu)\sqrt{1-\nu}}\sum_{i=1}^{d}\|g_{1:T,i}\|. \tag{26}
$$

The assumption $\frac{s_{t+1,i}}{\lambda_{t+1}} \geq \frac{s_{t,i}}{\lambda_t}$ is crucial as discussed in Ref. [3]. In two ways we can make sure that this assumption is satisfied: one is to increase ν gradually towards 1 during training, as proven in Ref. [3]; another is to define an AMSGRAD version of the adaptive method by substituting $s_{t,i} = \max(\sqrt{\frac{n_{t,i}}{c_n}}, \sqrt{\frac{n_{t-1,i}}{c_n}})$. However, the AMSGRAD "fix" is found to be harmful to the performance, and we discourage using an AMSGRAD version of LAPROP unless the problem demands so, especially at an early stage of training. Empirically, one may also enlarge the batch size to suppress the fluctuation of $s_{t,i}$ so that better convergence can be obtained.

B Practical Concerns and Various Extensions

B.1 Tuning ϵ

The tuning of ϵ is found to be important for stability and convergence of ADAM in some difficult tasks, such as in Rainbow DQN and Transformer training [29, 30], and a small ϵ may result in an unstable learning trajectory. However, we find that this is usually not the case for LAPROP, and LAPROP almost always works well with a small ϵ. The ϵ can be freely set to be 10^{-8}, 10^{-15} or 10^{-20}, as long as it is within the machine precision. On the other hand, if we consider ϵ as a term that helps the optimizer to converge in the presence of the assumption discussed in the last section, it can be tuned to a non-negligible value such as 10^{-4}, which actually, slows down the optimization significantly. Therefore, we encourage to use a larger batch size, a larger ν or an AMSGRAD version of LAPROP to ensure final convergence, rather than to use a large ϵ.

As LAPROP is considerably stable, we find that gradient clipping is unnecessary, and we do not use gradient clipping in our experiments.

B.2 Tuning ν

The tuning of ν is nontrivial and it is basically determined by trial and error. Generally, a smaller ν makes LAPROP closer to the signed gradient methods and the maximal update made by LAPROP becomes smaller, which may be beneficial in noisy settings, and a larger ν makes n_t change more slowly, which may benefit fine-tuning of the model and the final convergence. From a statistical viewpoint, ν is used to estimate the second moment of the gradient on sampled data, and therefore if the model changes quickly with optimization updates, a large ν will result in a large bias in the second-moment estimation. In our experiments, we find that a smaller ν sometimes indeed improves the loss faster at the initial stage where the model changes quickly, while a larger ν improves faster later, such as in Fig. 8b. For MNIST and CIFAR10, a small ν trains slightly faster only for the initial hundreds of updates and this occurs only for a limited range of ν, which may be due to the simplicity of the tasks. We notice that when the training is stable, a larger ν almost always makes the training faster, and therefore we believe that increasing ν to a larger value at a later stage is beneficial. The tuning of ν for large-scale and difficult tasks can potentially bring improved results and we leave it for future research.

B.3 Weight Decay for LaProp: LaPropW

As suggested in Ref. [5], we implement the weight decay separately, which may also be called the LAPROPW algorithm. The algorithm is given by Algorithm 2. Note that we combine the learning rate and the momentum so that even in the presence of a changing learning rate, the momentum still represents the average of the different update steps.

Algorithm 2 LAPROPW

Input: $\theta_1 \in \mathbb{R}^d$, learning rate $\{\lambda_t\}_{t=1}^T$, weight decay $\{w_t\}_{t=1}^T$, decay parameters $0 \le \mu < 1,\ 0 \le \nu < 1,\ \epsilon \ll 1$, bias correction factors $0 < c_n, c_m < 1$. Set $m_0 = 0,\ n_0 = 0$.

for $t = 1$ **to** T **do**

$\quad g_t = \nabla_\theta \ell(\theta_t)$

$\quad n_t = \nu n_{t-1} + (1 - \nu)g_t^2$

$\quad m_t = \mu m_{t-1} + \lambda_t(1 - \mu)\frac{g_t}{\sqrt{n_t/c_n}+\epsilon}$

$\quad \theta_{t+1} = (\theta_t - m_t/c_m) \times (1 - w_t)$

end for

B.4 AmsProp

In Ref. [3], the authors proposed AMSGRAD as a variant of ADAM that has a monotonically increasing n_t to guarantee its convergence. This idea may be applied to LAPROP similarly, which produces an algorithm that may be called *AmsProp*, as Algorithm 3. It should be noted that this algorithm subtly differs from the original AMSGRAD. In practical cases, we have not observed the advantage of using such a variant, because a large ν can often do well enough by approximately producing a constant n_t at convergence and therefore achieving a good performance. Nevertheless, AmsProp might be useful in special or complicated cases and we leave it for future research.

Algorithm 3 AmsProp

Input: $\theta_1 \in \mathbb{R}^d$, learning rate $\{\lambda_t\}_{t=1}^T$, decay parameters $0 \le \mu < 1$, $0 \le \nu < 1$, $\epsilon \ll 1$, bias correction factors $0 < c_n, c_m < 1$. Set $m_0 = 0$, $n_0 = 0$, $\tilde{n}_0 = 0$.

for $t = 1$ **to** T **do**

$\quad g_t = \nabla_\theta \ell(\theta_t)$

$\quad n_t = \nu n_{t-1} + (1 - \nu) g_t^2$

$\quad \tilde{n}_t = \max(\tilde{n}_{t-1}, n_t)$

$\quad m_t = \mu m_{t-1} + \lambda_t (1 - \mu) \frac{g_t}{\sqrt{\tilde{n}_t/c_n} + \epsilon}$

$\quad \theta_{t+1} = \theta_t - m_t/c_m$

end for

B.5 Centered LaProp

Following the suggestion in Ref. [13], we also propose a centered version of LAPROP, which uses the estimation of the centered second moment rather than the non-centered second moment, which is a more aggressive strategy that would diverge in the presence of a constant gradient. The algorithm is given by Algorithm 4. As the estimation of the centered momentum is unstable at the initial steps, one may not do the update for the initial steps or one may use the original LAPROP for the initial steps.

Also, one can even combine the centered LAPROP and AMSGRAD, using the maximum of the centered second moment of the gradient. However, we have not been aware of any advantage of such a combination.

Algorithm 4 Centered LAPROP

Input: $\theta_1 \in \mathbb{R}^d$, learning rate $\{\lambda_t\}_{t=1}^T$, decay parameters $0 \le \mu < 1$, $0 \le \nu < 1$, $\epsilon \ll 1$, bias correction factors $0 < c_n, c_m < 1$, initial centered update step $t_{\text{init}} > 1$. Set $m_0 = 0$, $n_0 = 0$, $\bar{n}_0 = 0$.

for $t = 1$ **to** T **do**

$\quad g_t = \nabla_\theta \ell(\theta_t)$

$\quad n_t = \nu n_{t-1} + (1 - \nu) g_t^2$

$\quad \bar{n}_t = \nu \bar{n}_{t-1} + (1 - \nu) g_t$

$\quad$ **if** $t \ge t_{\text{init}}$ **then**

$\quad\quad m_t = \mu m_{t-1} + \lambda_t (1 - \mu) \frac{g_t}{\sqrt{(n_t - \bar{n}_t^2)/c_n} + \epsilon}$

$\quad$ **else**

$\quad\quad m_t = m_{t-1} \quad$ or $\quad m_t = \mu m_{t-1} + \lambda_t (1 - \mu) \frac{g_t}{\sqrt{n_t/c_n} + \epsilon}$

$\quad$ **end if**

$\quad \theta_{t+1} = \theta_t - m_t/c_m$

end for

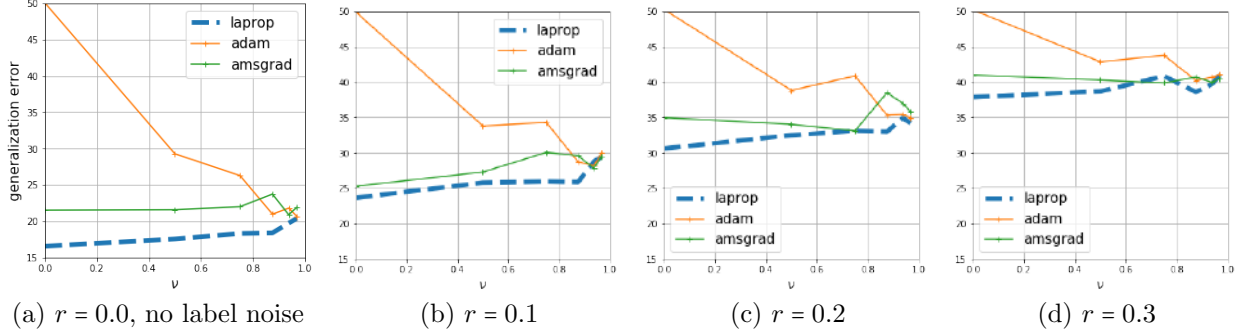| (a) $r = 0.0$, no label noise | (b) $r = 0.1$ | (c) $r = 0.2$ | (d) $r = 0.3$ |

Figure 7: Generalization error on label-corrupted IMDB plotted against ν, with flip probability r of labels. For all the four corruption rates, the behaviors of LAPROP are similar: a smaller ν gives a lower generalization error and better performance. For ADAM, the performance gets worse for a smaller ν due to divergence and instability; for AMSGRAD, the divergence is fixed and the learning is more stable than ADAM, but its failure to decouple μ and ν seems to have a negative effect on performance when compared with LAPROP.

C   Additional Experiments and Experimental Details

We use the Pytorch library [31] for deep learning, and the codes that are used to produce our results are released on Github[3].

C.1   Achieving Better Generalization Performance

While our analysis focuses on optimization, in this section we show how ν might be tuned to improve generalization. We consider the IMDB dataset, a binary classification task for which the goal is to identify the sentiment of the speakers, and it is a dataset where overfitting occurs quickly. We use LSTM [32] with pretrained GloVe word embedding [33], and we also study how label noise affects the different optimizers in this task. The label of every data point is randomly flipped with probability $r \in \{0, 0.1, 0.2, 0.3\}$.

The training batch size is 128 and the learning rate is set to 2×10^{-3} with $\mu = 0.9$. All the models are trained for 15 epochs. As shown in Figure 7, we find that (1) LAPROP with $\nu = 0$ always achieves the best generalization both with and without label noise, and especially at $r = 0$, LAPROP gives a 10% accuracy improvement over the best possible hyperparameter setting of ADAM and AMSGRAD; (2) ADAM and AMSGRAD are not very stable w.r.t the changes in ν, while LAPROP responds to the changes in ν in a stable and predictable way, implying that LAPROP's hyperparameter ν is easier to tune in practice.

C.2   Optimization of the Noisy Rosenbrock Loss

Most of the settings of the task are already described in the main text. To complete the details, the learning rate is set to be 10^{-2}, and we use $\mu = 0.9$ and $\epsilon = 10^{-8}$ for all the optimizers.

C.3   Neural Style Transfer

We closely follow the prescription given in the Pytorch tutorial on neural transfer.[4] A pretrained VGG19 network is used to extract the features of a style image and a content image, and we optimize

---

[3]`https://github.com/Z-T-WANG/LaProp-Optimizer`
[4]`https://pytorch.org/tutorials/advanced/neural_style_tutorial.html`

an input image so that its style follows the style image and its content follows the content image. The loss associated to content is calculated using the features extracted at the 4th convolutional layer of the network, and the loss associated to style is calculated as the average from the 1st to the 5th layer, using the normalized Gram matrices. All the settings follow the example given by Pytorch, except for the settings concerning the optimizers and that a black empty image is used as our input image. A learning rate of 10^{-2} is used, and we use $\mu = 0.9$ and $\epsilon = 10^{-15}$, both for ADAM and LAPROP, and for SGD we still use these parameters. Note that this optimization task is complex and usually L-BFGS is used as the optimizer.

C.4   Training Extremely Deep Fully Connected Networks

The network has input dimension 784, followed by d layers all of which include w hidden neurons, then followed by an output layer of dimension 10. The training batch size is 1024, and the experiment includes 400 epochs. For all the tasks, we set $\mu = 0.8$, $\nu = 0.96, \epsilon = 10^{-26}$, and we only tune for a proper learning rate, ranging from $\lambda = 4 \times 10^{-4}$ to 10^{-5}. To plot the training curves, we have applied smoothing over consecutive 50 updates.

C.5   Translation and Pretraining with Transformers

For the IWSLT task, we use the default settings provided by Ref. [28] except for the learning rate schedule. As shown in Ref. [4], if we use a maximum learning rate of 3×10^{-4}, ADAM will be trapped in a bad local minimum unless we use a warmup. Therefore, we use the 3×10^{-4} learning rate to reproduce this result for ADAM, and we also use it to test LAPROP. After the warmup, or if no warmup is used, the learning rate linearly decreases and vanishes at the 60×10^3-th update. Interestingly, we find that the result shown in the main text is highly learning-rate-dependent: if we use a maximum learning rate of 10^{-4} rather than 3×10^{-4}, ADAM will not get trapped in a bad minimum and both LAPROP and ADAM can work well without a warmup schedule; if we use a maximum learning rate of 10^{-3}, LAPROP will not show a large difference from ADAM as in the main text. Therefore, the advantage provided by LAPROP does not remove the necessity of a warmup schedule, and the result in the main text is only for demonstrating the properties of LAPROP. To plot the learning curves, we record the raw training loss of all update steps and apply a Gaussian smoothing with a standard deviation of 70 to smooth the data points.

For the RoBERTa task, the training batch size is 120 and the maximum learning rate is 10^{-4}, and the learning rate linearly decreases and is expected to vanish at the 125×10^3-th update, although we only carry out the initial 20×10^3 updates. We use the momentum $\mu = 0.9$ and the default RoBERTa setting $\epsilon = 10^{-6}$ for ADAM and the default LAPROP setting $\epsilon = 10^{-15}$ for LAPROP. The maximum sequence length of data is kept 512 as default, and the trained model is Bert-base [30], as provided by the Fairseq library [28]. We also enable the option of mask-whole-words in the Fairseq library to make the pretraining task more realistic.

We conjecture that the unnecessity of a warmup is due to the small learning rate and the relatively small variance of the training data, i.e. only containing the English Wikipedia. In this case, we notice that a large ν may be used. We present the learning curves of different ν and using or not using warmup with more details in Fig. 8, training for the initial 20×10^3 updates, which is actually less than one epoch. If one zooms in Fig. 8a, it can also be observed that LAPROP marginally outperforms ADAM from an early stage. In Fig. 8b, it can be seen that a smaller ν

(a)	(b)
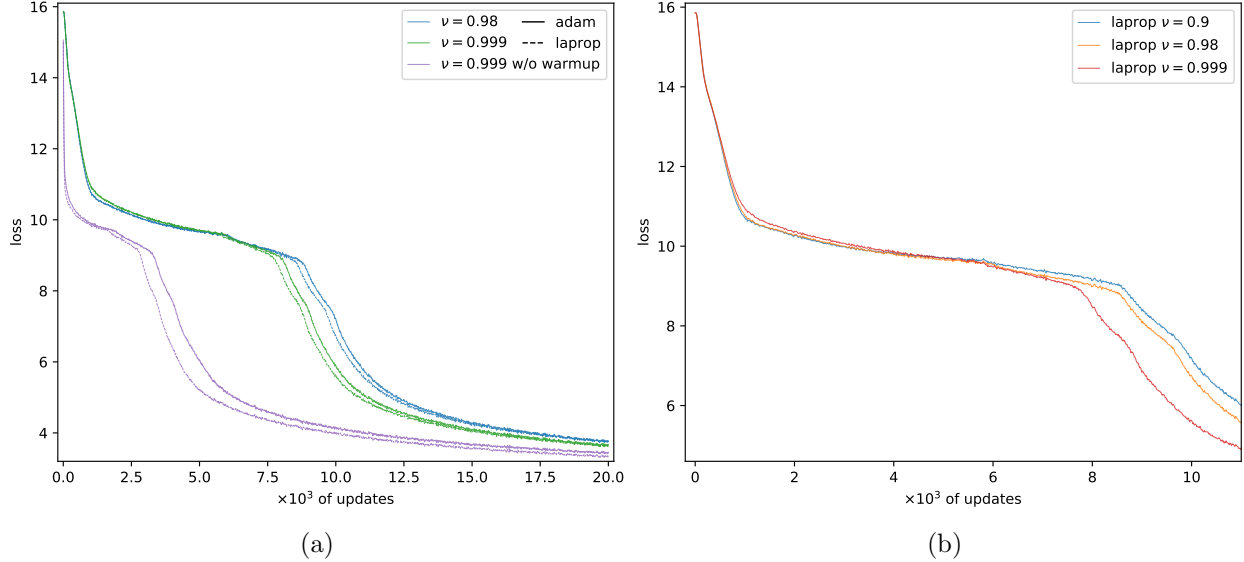
Figure 8: RoBERTa pretraining on the full English Wikipedia. See the text for details.

accelerate the training at an early stage, while a larger ν converges better at a later stage. We have applied a Gaussian smoothing with a standard deviation $\sigma = 6$ to the nearby data points in the plots, which is the same as in the main text. The fluctuation of the loss is much smaller than the case of IWSLT.

The English Wikipedia dataset is prepared following the instructions in the GitHub repository of GluonNLP [34].[5] We used the latest English Wikipedia dump file and cleaned the text,[6] and we encode and preprocess the text following the standard RoBERTa procedure.[7]

C.6   Details of Reinforcement Learning Experiments

The curves of training performance of the 20 Atari2600 games in our experiments are given in Fig. 9. The random seeds for LaProp and Adam are always the same, so that the randomness in initialization and in the game process are removed. We can see that LaProp makes progress earlier than Adam.

To obtain faster training results, compared with Rainbow DQN [29], we change the update period of target networks to 10000, and we avoid stop or loop of the games by forcing the agent to take random actions to lose a life if 10000 frames have passed in an episode. We also use a combined ϵ-greedy and noisy network strategy, and the memory replay buffer only contains 0.5 million transitions and the buffer adopts a random replace strategy when it becomes full. Concerning the noisy network strategy, the trained network and the target network always use the same noise on training data to enhance consistency in their evaluations, which may have affected the performance marginally. Specifically for Atari2600 game *Breakout*, we do not use multi-step

---

[5]https://github.com/dmlc/gluon-nlp/issues/641

[6]using scripts on https://github.com/eric-haibin-lin/text-proc

[7]described in https://github.com/pytorch/fairseq/blob/master/examples/roberta/README.pretraining.md
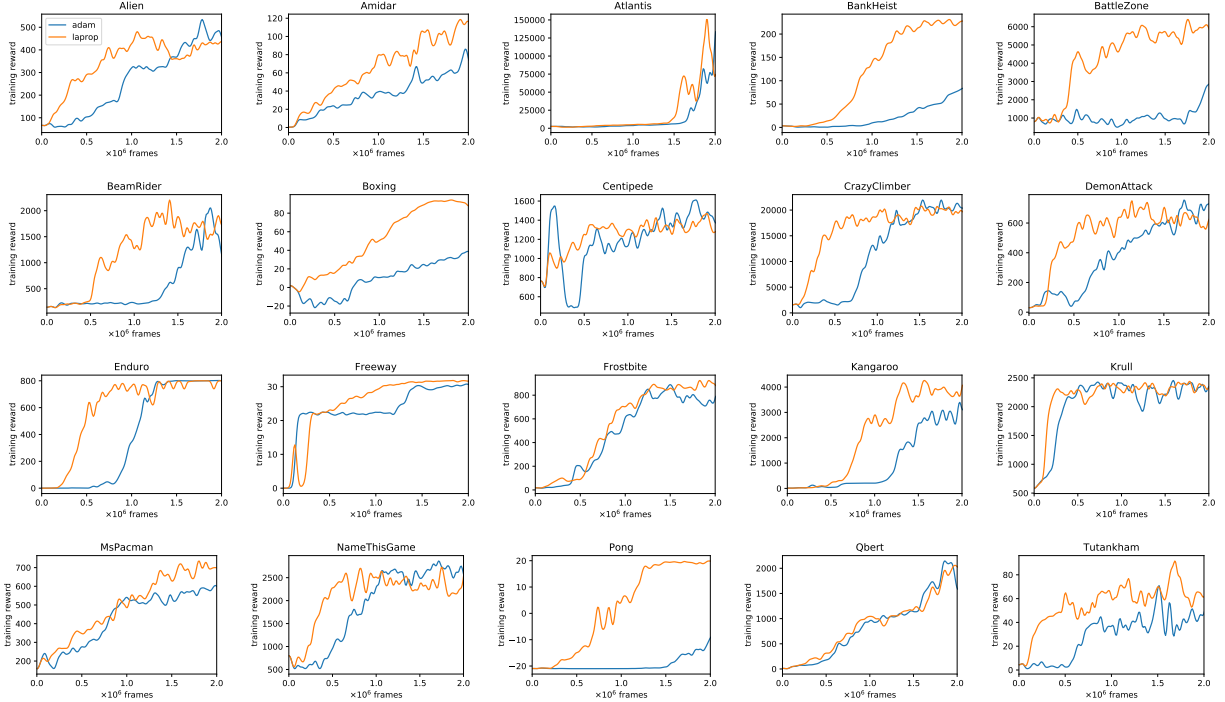
Figure 9: Curves of training performance on 20 Atari2600 games, where the blue curves represent ADAM and the orange curves represent LAPROP. The training starts at the 8×10^4-th frame, and per 10^4 frames we record the average reward for a life in the game as the data point of training performance, and we use a Gaussian smoothing with $\sigma = 2$ on the curves.

learning or the duel network structure, and the minimum value of ϵ in the greedy-ϵ strategy is reduced from 0.01 to 0.005.

Other parameter settings follow Ref. [29], except for the ϵ of LAPROP, which is still the LAPROP default $\epsilon = 10^{-15}$. Therefore, ϵ of ADAM is actually larger than that of LAPROP. It is worth mentioning that the training batch size is 32, which is quite small, and that we have $\mu = 0.9$ and $\nu = 0.999$, with a learning rate of 6.25×10^{-5}.

C.7   Image Classification on CIFAR10

We train deep residual networks on the CIFAR10 image classification task using LAPROP and ADAM [35, 36], implementing weight decay following the suggestions in Ref. [5]. The network architecture is the standard Resnet-20,[8] and we use the same hyperparameters for ADAM and LAPROP except for ϵ that is set to their default, and we perform a grid search on μ and ν. The results on test accuracy are shown in Fig. 10. We find that LAPROP and ADAM perform comparably around the region of common hyperparameter choices, and when ν gets smaller ADAM occasionally diverges, and if μ is much larger than ν then ADAM diverges, while LAPROP is not affected. We believe that the residual connections have made the model stable and more robust against divergence in this task. Interestingly, when we only look at the loss, we find that ADAM tends to overfit when ν is

---

[8]We use the implementation in `https://github.com/bearpaw/pytorch-classification`

adam — μ vs ν

$\mu \backslash \nu$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
0.80	nan	91.3	91.3	91.1	91.5	91.6	91.4	91.2	91.3	91.5	91.4
0.85	nan	91.1	91.5	91.6	91.8	91.7	91.6	91.4	91.6	91.4	91.2
0.90	nan	91.3	91.7	10.0	91.6	90.9	91.7	91.2	91.4	91.5	91.4
0.95	nan	nan	91.4	91.4	91.3	91.4	91.3	91.3	91.4	91.5	91.1
0.99	nan	nan	nan	nan	10.0	nan	nan	10.0	10.0	91.2	91.3

laprop — μ vs ν

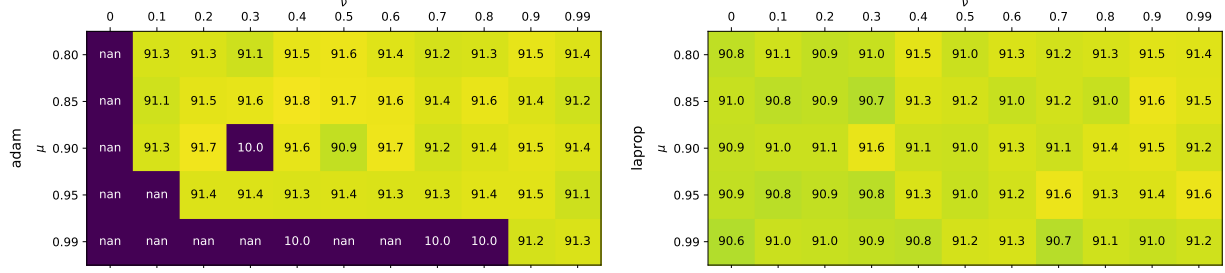| $\mu \backslash \nu$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
0.80	90.8	91.1	90.9	91.0	91.5	91.0	91.3	91.2	91.3	91.5	91.4
0.85	91.0	90.8	90.9	90.7	91.3	91.2	91.0	91.2	91.0	91.6	91.5
0.90	90.9	91.0	91.1	91.6	91.1	91.0	91.3	91.1	91.4	91.5	91.2
0.95	90.9	90.8	90.9	90.8	91.3	91.0	91.2	91.6	91.3	91.4	91.6
0.99	90.6	91.0	91.0	90.9	90.8	91.2	91.3	90.7	91.1	91.0	91.2

Figure 10: Test accuracy (%) of Resnet-20 trained on CIFAR10, corresponding to the grid search models in Fig. 12. *NAN* or *nan* is an abbreviation for *not a number*, which means that the machine encounters a numerical problem such as numerical overflow, and that it cannot obtain a number.

adam (training loss) — μ vs ν

$\mu \backslash \nu$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
0.80	NAN	0.028	0.027	0.03	0.029	0.028	0.026	0.029	0.028	0.026	0.026
0.85	NAN	0.031	0.027	0.028	0.025	0.027	0.026	0.028	0.029	0.027	0.026
0.90	NAN	0.026	0.027	2.3	0.028	0.027	0.027	0.027	0.028	0.026	0.026
0.95	NAN	NAN	0.026	0.027	0.026	0.029	0.027	0.029	0.028	0.028	0.027
0.99	NAN	NAN	NAN	NAN	>100	NAN	NAN	>100	>100	0.032	0.028

laprop (training loss) — μ vs ν

$\mu \backslash \nu$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
0.80	0.077	0.059	0.055	0.052	0.045	0.044	0.042	0.038	0.032	0.029	0.027
0.85	0.075	0.058	0.056	0.055	0.048	0.046	0.042	0.037	0.035	0.03	0.027
0.90	0.073	0.061	0.055	0.051	0.048	0.046	0.039	0.04	0.033	0.03	0.028
0.95	0.078	0.059	0.06	0.053	0.05	0.049	0.046	0.038	0.036	0.029	0.027
0.99	0.072	0.065	0.061	0.054	0.055	0.051	0.046	0.045	0.038	0.034	0.03

(a) The training loss on CIFAR10 of ADAM and LAPROP

adam (test loss) — μ vs ν

$\mu \backslash \nu$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
0.80	NAN	0.38	0.37	0.37	0.36	0.38	0.38	0.38	0.36	0.35	0.35
0.85	NAN	0.38	0.38	0.37	0.36	0.36	0.35	0.36	0.36	0.36	0.36
0.90	NAN	0.39	0.37	2.3	0.36	0.38	0.36	0.35	0.36	0.36	0.35
0.95	NAN	NAN	0.39	0.38	0.37	0.36	0.36	0.36	0.36	0.34	0.37
0.99	NAN	NAN	NAN	NAN	>100	NAN	NAN	3.2	>100	0.35	0.35

laprop (test loss) — μ vs ν

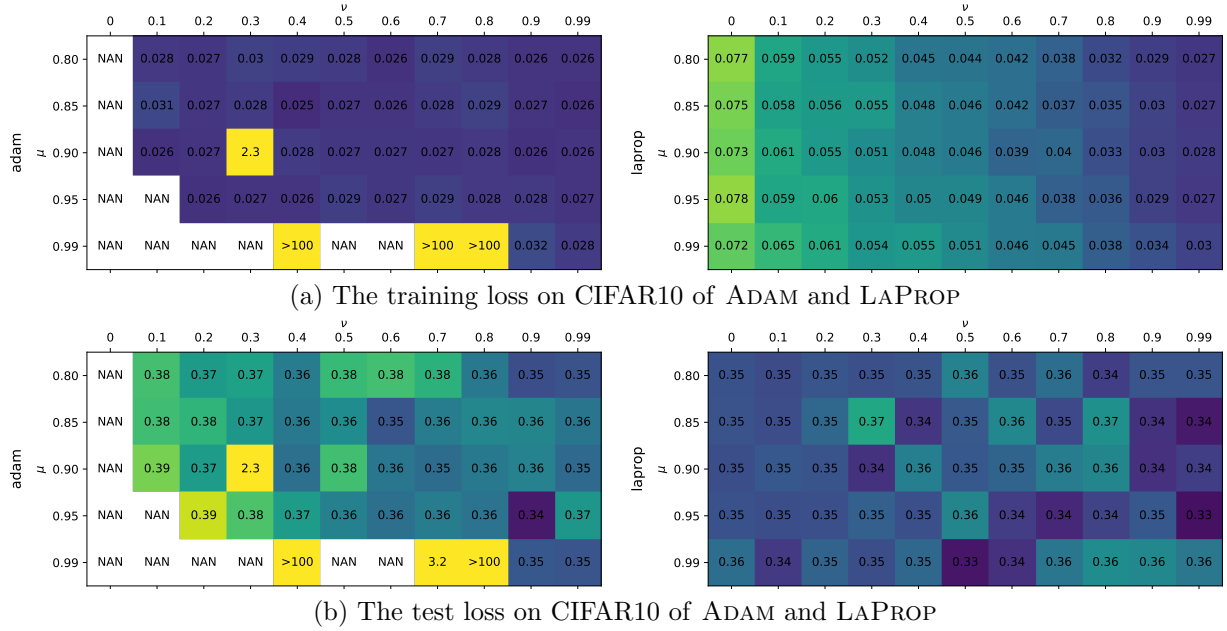| $\mu \backslash \nu$	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	0.99
0.80	0.35	0.35	0.35	0.35	0.35	0.36	0.35	0.36	0.34	0.35	0.35
0.85	0.35	0.35	0.35	0.37	0.34	0.35	0.36	0.35	0.37	0.34	0.34
0.90	0.35	0.35	0.35	0.34	0.36	0.35	0.35	0.36	0.36	0.34	0.34
0.95	0.35	0.35	0.35	0.35	0.35	0.36	0.34	0.34	0.34	0.35	0.33
0.99	0.36	0.34	0.35	0.35	0.35	0.33	0.34	0.36	0.36	0.36	0.36

(b) The test loss on CIFAR10 of ADAM and LAPROP

Figure 11: A summary of the final training and test loss in Fig. 12. From the trend, we can clearly see that ADAM tends to overfit in terms of loss when it is closer to divergence, while LAPROP is not clearly affected. We see that if ADAM does not diverge, ADAM always reaches a low training loss irrespective of μ and ν, while the training loss of LAPROP is clearly dependent on μ and ν. However, the test loss of LAPROP is almost unaffected, but the test loss of ADAM increases when it is closer to divergence, as shown for $\mu \leq 0.95$ and $0.1 \leq \nu \leq 0.4$, where the loss is higher than that of LAPROP. This is an example where LAPROP generalizes better than ADAM. Surprisingly, we find that the higher test loss of ADAM does not necessarily result in a worse test accuracy, which we think is probably a phenomenon specific to this task.

small while LAPROP does not, as shown in Fig. 11 and discussed in the caption. We have yet to find an explanation for this phenomenon.

We use a learning rate of 10^{-3} and a weight decay of 10^{-4}, and they are both reduced by a factor of 10 at epoch 80 and 120, and we train for 164 epochs and report the average of the last 5 epochs. Other settings follow Ref. [35]. The complete learning curves of training and test loss are given in Fig. 12.
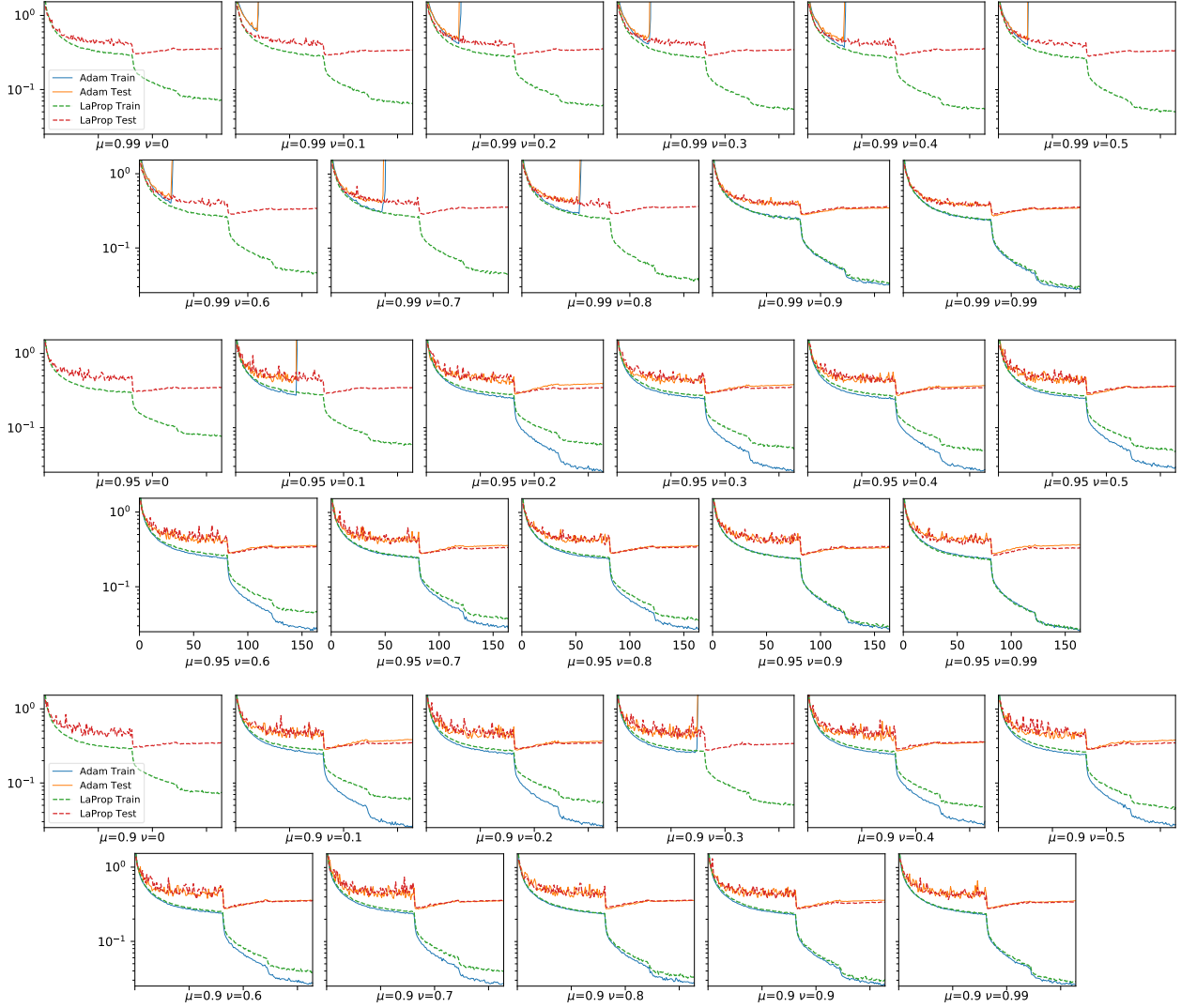
Figure 12: The learning curves of the Resnet-20 grid search on CIFAR10. The training loss and test loss are plotted for ADAM and LAPROP, and the meaning of the curves are shown in the legend in the first plot. If ADAM diverges, its curves become absent in the plots. In the above figures, we see that when ADAM diverges, a smaller ν causes the divergence to occur earlier. However, divergence sometimes occurs accidentally, such as the case of $\mu = 0.9, \nu = 0.3$. We see that the curves of LAPROP and ADAM resemble, except for that ADAM often reaches a lower training loss. For the rest of the figures see Fig. 13.
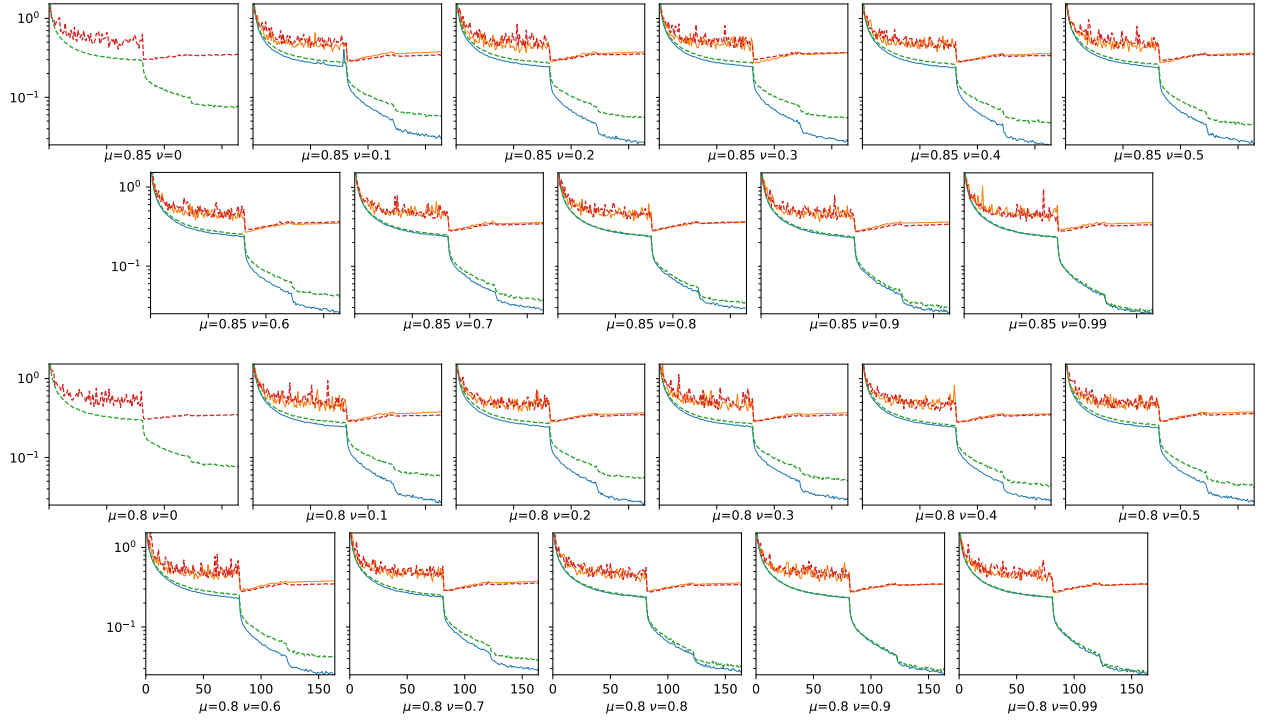
Figure 13

26