No More Pesky Learning Rates

Tom Schaul SCHAUL@CIMS.NYU.EDU
Sixin Zhang ZSX@CIMS.NYU.EDU
Yann LeCun YANN@CIMS.NYU.EDU
Courant Institute of Mathematical Sciences
New York University
715 Broadway, New York, NY 10003, USA

Abstract

The performance of stochastic gradient descent (SGD) depends critically on how learning rates are tuned and decreased over time. We propose a method to automatically adjust multiple learning rates so as to minimize the expected error at any one time. The method relies on local gradient variations across samples. In our approach, learning rates can increase as well as decrease, making it suitable for non-stationary problems. Using a number of convex and non-convex learning tasks, we show that the resulting algorithm matches the performance of SGD or other adaptive approaches with their best settings obtained through systematic search, and effectively removes the need for learning rate tuning.

1. Introduction

Large-scale learning problems require algorithms that scale benignly (e.g. sub-linearly) with the size of the dataset and the number of trainable parameters. This has lead to a recent resurgence of interest in *stochastic gradient descent* methods (SGD). Besides fast convergence, SGD has sometimes been observed to yield significantly better generalization errors than batch methods (Bottou & Bousquet, 2011).

In practice, getting good performance with SGD requires some manual adjustment of the initial value of the learning rate (or step size) for each model and each problem, as well as the design of an annealing schedule for stationary data. The problem is particularly acute for non-stationary data.

The contribution of this paper is a novel method to *automatically adjust* learning rates (possibly different learning rates for different parameters), so as to minimize some estimate of the expectation of the loss at any one time.

Starting from an idealized scenario where every sample's contribution to the loss is quadratic and separable, we derive a formula for the optimal learning rates for SGD, based on estimates of the variance of the gradient. The formula has two components: one that captures variability across samples, and one that captures the local curvature, both of which can be estimated in practice. The method can be used to derive a single common learning rate, or local learning rates for each parameter, or each block of parameters, leading to five variations of the basic algorithm, none of which need any parameter tuning.

The performance of the methods *obtained without any manual tuning* are reported on a variety of convex and non-convex learning models and tasks. They compare favorably with an "ideal SGD", where the best possible learning rate was obtained through systematic search, as well as previous adaptive schemes.

2. Background

SGD methods have a long history in adaptive signal processing, neural networks, and machine learning, with an extensive literature (see (Bottou, 1998; Bottou & Bousquet, 2011) for recent reviews). While the practical advantages of SGD for machine learning applications have been known for a long time (LeCun et al., 1998), interest in SGD has increased in recent years due to the ever-increasing amounts of streaming data, to theoretical optimality results for generalization error (Bottou & LeCun, 2004), and to competitions being won by SGD methods, such as the PASCAL Large Scale Learning Challenge (Bordes et al., 2009), where Quasi-Newton approximation of the Hessian was used within SGD. Still, practitioners need to deal with a sensitive hyper-parameter tuning phase to get top performance: each of the PASCAL tasks used

Proceedings of the 30^{th} International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

very different parameter settings. This tuning is very costly, as every parameter setting is typically tested over multiple epochs.

Learning rates in SGD are generally decreased according a schedule of the form $\eta(t) = \eta_0(1 + \gamma t)^{-1}$. Originally proposed as $\eta(t) = O(t^{-1})$ in (Robbins & Monro, 1951), this form was recently analyzed in (Xu, 2011; Bach & Moulines, 2011) from a non-asymptotic perspective to understand how hyper-parameters like η_0 and γ affect the convergence speed.

Numerous researchers have proposed schemes for making learning rates adaptive, either globally or by adapting one rate per parameter ('diagonal preconditioning'); see (George & Powell, 2006) for an overview. An early diagonal preconditioning schemes was proposed in (Almeida & Langlois, 1999) where the learning rate is adapted as

$$\eta_i(t) = \max\left(0, \frac{\eta_0\ \theta_i(t) \cdot \nabla_{\theta_i}^{(t-1)}}{\overline{v_i}}\right)$$

for each problem dimension i, where $\nabla_{\theta_i}^{(t)}$ is gradient of the ith parameter at iteration t, and $\overline{v_i} \approx \mathbb{E}\left[\nabla_{\theta_i}^2\right]$ is a recent running average of its square. Stochastic meta-descent (SMD, Schraudolph (1999; 2002)) uses a related multiplicative update of learning rates. Approaches based on the natural gradient (Amari et al., 2000) precondition the updates by the empirical Fisher information matrix (estimated by the gradient covariance matrix, or its diagonal approximation), in the simplest case: $\eta_i = \eta_0/\overline{v_i}$; the "Natural Newton" algorithm (Le Roux & Fitzgibbon, 2010) combines the gradient covariance with second-order information. Finally, derived from a worst-case analysis, (Duchi et al., 2010) propose an approach called 'AdaGrad', where the learning rate takes the form

$$\eta_i(t) = \frac{\eta_0}{\sqrt{\sum_{s=0}^{t}\left(\nabla_{\theta_i}^{(s)}\right)^2}}.$$

The main practical drawback for all of these approaches is that they retain one or more sensitive hyper-parameters, which must be tuned to obtain satisfactory performance. AdaGrad has another disadvantage: because it accumulates all the gradients from the moment training starts to determine the current learning rate, the learning rate monotonically decreases: this is especially problematic for non-stationary problems, but also on stationary ones, as navigating the properties of optimization landscape change continuously.

The main contribution of the present paper is *a formula that gives the value of the learning rate that will maximally decrease the expected loss after the next update*. For efficiency reasons, some terms in the formula must be approximated using such quantities as the mean and variance of the gradient. As a result, the learning rate is *automatically decreased to zero* when approaching an optimum of the loss, without requiring a pre-determined annealing schedule, and if the problem is non-stationary, it the learning rate *grows again* when the data changes.

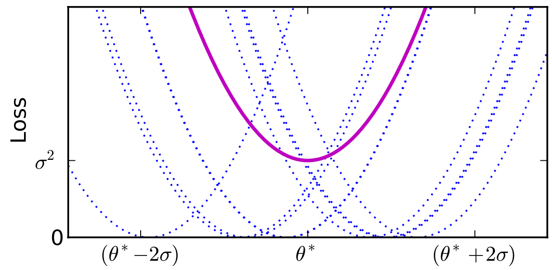

Figure 1. Illustration of the idealized loss function considered (thick magenta), which is the average of the quadratic contributions of each sample (dotted blue), with minima distributed around the point θ^*. Note that the curvatures are assumed to be identical for all samples.

3. Optimal Adaptive Learning Rates

In this section, we derive an optimal learning rate schedule, using an idealized quadratic and separable loss function. We show that using this learning rate schedule preserves convergence guarantees of SGD. In the following section, we find how the optimal learning rate values can be estimated from available information, and describe a couple of possible approximations.

The samples, indexed by j, are drawn i.i.d. from a data distribution $\mathcal{P}$. Each sample contributes a *per-sample loss* $\mathcal{L}^{(j)}(\boldsymbol{\theta})$ to the expected loss:

$$\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{j\sim\mathcal{P}}\left[\mathcal{L}^{(j)}(\boldsymbol{\theta})\right] \tag{1}$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is the trainable parameter vector, whose optimal value is denoted $\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$. The SGD parameter update formula is of the form $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta^{(t)}\nabla_{\boldsymbol{\theta}}^{(j)}$, where $\nabla_{\boldsymbol{\theta}}^{(j)} = \frac{\partial}{\partial\boldsymbol{\theta}}\mathcal{L}^{(j)}(\boldsymbol{\theta})$ is the gradient of the the contribution of example j to the loss, and the learning rate $\eta^{(t)}$ is a suitably chosen sequence of positive scalars (or positive definite matrices).

3.1. Noisy Quadratic Loss

We assume that the per-sample loss functions are smooth around minima, and can be locally approximated by a quadratic function. We also assume that the minimum value of the per-sample loss functions are zero:

$$\mathcal{L}^{(j)}(\boldsymbol{\theta}) = \frac{1}{2}\left(\boldsymbol{\theta} - \mathbf{c}^{(j)}\right)^\top \mathbf{H}^{(j)}\left(\boldsymbol{\theta} - \mathbf{c}^{(j)}\right)$$
$$\nabla_{\boldsymbol{\theta}}^{(j)} = \mathbf{H}^{(j)}\left(\boldsymbol{\theta} - \mathbf{c}^{(j)}\right)$$

where $\mathbf{H}_i$ is the (positive semi-definite) Hessian matrix of the per-sample loss of sample j, and $\mathbf{c}^{(j)}$ is the optimum for that sample. The distribution of per-sample optima $\mathbf{c}^{(j)}$ has mean $\boldsymbol{\theta}^*$ and variance $\boldsymbol{\Sigma}$. Figure 1 illustrates the scenario in one dimension.

To simplify the analysis, we assume for the remainder of this section that the Hessians of the per-sample losses are identical for all samples, and that the problem is separable, i.e., the Hessians are diagonal, with diagonal terms denoted $\{h_1, \ldots, h_i, \ldots, h_d\}$. Further, we will ignore the off-diagonal terms of $\boldsymbol{\Sigma}$, and denote the diagonal $\{\sigma_1^2, \ldots, \sigma_i^2, \ldots, \sigma_d^2\}$. Then, for any of the d dimensions, we thus obtain a one-dimensional problem (all indices i omitted).

$$J(\theta) = \mathbb{E}_{i\sim\mathcal{P}}\left[\frac{1}{2}h(\theta - c^{(j)})^2\right] = \frac{1}{2}h\left[(\theta-\theta^*)^2 + \sigma^2\right] \quad (2)$$

The gradient components are $\nabla_\theta^{(j)} = h\left(\theta - c^{(j)}\right)$, with

$$\mathbb{E}[\nabla_\theta] = h(\theta - \theta^*) \qquad \mathbb{V}\text{ar}[\nabla_\theta] = h^2\sigma^2 \quad (3)$$

and we can rewrite the SGD update equation as

$$\begin{aligned}\theta^{(t+1)} &= \theta^{(t)} - \eta h\left(\theta^{(t)} - c^{(j)}\right)\\ &= (1-\eta h)\theta^{(t)} + \eta h\theta^* + \eta h\sigma\xi^{(j)} \quad (4)\end{aligned}$$

where the $\xi^{(j)}$ are i.i.d. samples from a zero-mean and unit-variance Gaussian distribution. Inserting this into equation 2, we obtain the expected loss after an SGD update

$$\begin{aligned}&\mathbb{E}\left[J\left(\theta^{(t+1)}\right) \mid \theta^{(t)}\right]\\ =\ & \frac{1}{2}h\cdot\left[(1-\eta h)^2(\theta^{(t)}-\theta^*)^2 + \eta^2h^2\sigma^2 + \sigma^2\right]\end{aligned}$$

3.2. Greedy Optimality

We can now derive the optimal (greedy) learning rates for the current time t as the value $\eta^*(t)$ that minimizes the expected loss after the next update

$$\begin{aligned}\eta^*(t) &= \arg\min_\eta\left[(1-\eta h)^2(\theta^{(t)}-\theta^*)^2 + \sigma^2 + \eta^2h^2\sigma^2\right]\\ &= \arg\min_\eta\Big[\eta^2\left(h(\theta^{(t)}-\theta^*)^2 + h\sigma^2\right)\\ &\qquad -2\eta(\theta^{(t)}-\theta^*)^2\Big]\\ &= \frac{1}{h}\cdot\frac{(\theta^{(t)}-\theta^*)^2}{(\theta^{(t)}-\theta^*)^2+\sigma^2} \quad (5)\end{aligned}$$

In the classical (noiseless or batch) derivation of the optimal learning rate, the best value is simply $\eta^*(t) = h^{-1}$. The above formula inserts a corrective term that *reduces* the learning rate whenever the sample pulls the parameter vector in different directions, as measured by the gradient variance σ^2. The reduction of the learning rate is larger near an optimum, when $(\theta^{(t)}-\theta^*)^2$ is small relative to σ^2. In effect, this will reduce the expected error due to the noise in the gradient. Overall, this will have the same effect as the usual method of progressively decreasing the learning rate as we get closer to the optimum, *but it makes this annealing schedule automatic.*

If we do gradient descent with $\eta^*(t)$, then almost surely, the algorithm converges (for the quadratic model). The proof is given in the appendix.

3.3. Global vs. Parameter-specific Rates

The previous subsections looked at the optimal learning rate in the one-dimensional case, which can be trivially generalized to d dimensions if we assume that all parameters are separable, namely by using an individual learning rate η_i^* for each dimension i. Alternatively, we can derive an optimal *global* learning rate η_g^* (see supplementary material for the full derivation),

$$\eta_g^*(t) \quad = \quad \frac{\sum_{i=1}^d h_i^2(\theta_i^{(t)}-\theta_i^*)^2}{\sum_{i=1}^d\left(h_i^3(\theta_i^{(t)}-\theta_i^*)^2 + h_i^3\sigma_i^2\right)} \quad (6)$$

which is especially useful if the problem is badly conditioned.

In-between a global and a component-wise learning rate, it is of course possible to have common learning rates for blocks of parameters. In the case of multilayer learning systems, the blocks may regroup the parameters of each single layer, the biases, etc. This is particularly useful in *deep* learning, where the gradient magnitudes can vary significantly between shallow and deep layers.

4. Approximations

In practice, we are not given the quantities σ_i, h_i and $(\theta_i^{(t)}-\theta_i^*)^2$. However, based on equation 3, we can estimate them from the observed samples of the gradient:

$$\eta_i^* = \frac{1}{h_i}\cdot\frac{(\mathbb{E}[\nabla_{\theta_i}])^2}{(\mathbb{E}[\nabla_{\theta_i}])^2 + \mathbb{V}\text{ar}[\nabla_{\theta_i}]} = \frac{1}{h_i}\cdot\frac{(\mathbb{E}[\nabla_{\theta_i}])^2}{\mathbb{E}[\nabla_{\theta_i}^2]} \quad (7)$$

The situation is slightly different for the global learning rate η_g^*. Here we assume that it is feasible to estimate the maximal curvature $h^+ = \max_i(h_i)$ (which can be done efficiently, for example using the diagonal Hessian computation method described in (LeCun et al., 1998)). Then we have the bound

$$\eta_g^*(t) \geq \frac{1}{h^+} \cdot \frac{\sum_{i=1}^d h_i^2(\theta_i^{(t)} - \mu_i)^2}{\sum_{i=1}^d \left(h_i^2(\theta_i^{(t)} - \mu_i)^2 + h_i^2\sigma_i^2\right)} = \frac{1}{h^+} \cdot \frac{\|\mathbb{E}[\nabla_{\boldsymbol{\theta}}]\|^2}{\mathbb{E}\left[\|\nabla_{\boldsymbol{\theta}}\|^2\right]} \quad (8)$$

because

$$\mathbb{E}\left[\|\nabla_{\boldsymbol{\theta}}\|^2\right] = \mathbb{E}\left[\sum_{i=1}^d (\nabla_{\theta_i})^2\right] = \sum_{i=1}^d \mathbb{E}\left[(\nabla_{\theta_i})^2\right]$$

In both cases (equations 7 and 8), the optimal learning rate is decomposed into two factors, one term which is the inverse curvature (as is the case for batch second-order methods), and one novel term that depends on the noise in the gradient, relative to the expected squared norm of the gradient. Below, we approximate these terms separately. For the investigations below, when we use the true values instead of a practical algorithm, we speak of the 'oracle' variant (e.g. in Figure 2, or Figure 1 of the supplementary material).

4.1. Approximate Variability

We use an exponential moving average with time-constant τ (the approximate number of samples considered from recent memory) for online estimates of the quantities in equations 7 and 8:

$$\begin{aligned}
\overline{g_i}(t+1) &= (1-\tau_i^{-1}) \cdot \overline{g_i}(t) + \tau_i^{-1} \cdot \nabla_{\theta_i(t)} \\
\overline{v_i}(t+1) &= (1-\tau_i^{-1}) \cdot \overline{v_i}(t) + \tau_i^{-1} \cdot (\nabla_{\theta_i(t)})^2 \\
\overline{l}(t+1) &= (1-\tau^{-1}) \cdot \overline{l}(t) + \tau^{-1} \cdot \|\nabla_{\boldsymbol{\theta}}\|^2
\end{aligned}$$

where $\overline{g_i}$ estimates the average gradient component i, $\overline{v_i}$ estimates the uncentered variance on gradient component i, and $\overline{l}$ estimates the squared length of the gradient vector:

$$\overline{g_i} \approx \mathbb{E}[\nabla_{\theta_i}] \qquad \overline{v_i} \approx \mathbb{E}[\nabla_{\theta_i}^2] \qquad \overline{l} \approx \mathbb{E}\left[\|\nabla_{\boldsymbol{\theta}}\|^2\right]$$

and we need $\overline{v_i}$ only for an element-wise adaptive learning rate and $\overline{l}$ only in the global case.

4.2. Adaptive Time-constant

We want the size of the memory to increase when the steps taken are small (increment by 1), and to decay quickly if a large step (close to the Newton step) is taken, which is obtained naturally, by the following update

$$\tau_i(t+1) = \left(1 - \frac{\overline{g_i}(t)^2}{\overline{v_i}(t)}\right) \cdot \tau_i(t) + 1,$$

Algorithm 1 Stochastic gradient descent with adaptive learning rates (element-wise, vSGD-l).

repeat
  draw a sample $c^{(j)}$, compute the gradient $\nabla_{\boldsymbol{\theta}}^{(j)}$, and compute the diagonal Hessian estimates $h_i^{(j)}$ using the "bbprop" method
  for $i \in \{1, \ldots, d\}$ **do**
    update moving averages

$$\begin{aligned}
\overline{g_i} &\leftarrow (1-\tau_i^{-1}) \cdot \overline{g_i} + \tau_i^{-1} \cdot \nabla_{\theta_i}^{(j)} \\
\overline{v_i} &\leftarrow (1-\tau_i^{-1}) \cdot \overline{v_i} + \tau_i^{-1} \cdot \left(\nabla_{\theta_i}^{(j)}\right)^2 \\
\overline{h_i} &\leftarrow (1-\tau_i^{-1}) \cdot \overline{h_i} + \tau_i^{-1} \cdot \left|\text{bbprop}(\boldsymbol{\theta})_i^{(j)}\right|
\end{aligned}$$

    estimate learning rate $\eta_i^* \leftarrow \frac{(\overline{g_i})^2}{\overline{h_i} \cdot \overline{v_i}}$
    update memory size
    $\tau_i \leftarrow \left(1 - \frac{(\overline{g_i})^2}{\overline{v_i}}\right) \cdot \tau_i + 1$
    update parameter $\theta_i \leftarrow \theta_i - \eta_i^* \nabla_{\theta_i}^{(j)}$
  end
until *stopping criterion is met*

This way of making the memory size adaptive allows us to eliminate one otherwise tuning-sensitive hyperparameter. Note that these updates (correctly) do not depend on the local curvature, making them scale-invariant.

4.3. Approximate Curvature

There exist a number of methods for obtaining an online estimates of the diagonal Hessian (Martens et al., 2012; Bordes et al., 2009; Chapelle & Erhan, 2011; Schaul & LeCun, 2013). We adopt the "bbprop" method, which computes positive estimates of the diagonal Hessian terms (Gauss-Newton approximation) for a single sample $h_i^{(j)}$, using a back-propagation formula (LeCun et al., 1998). The diagonal estimates are used in an exponential moving average procedure

$$\overline{h_i}(t+1) = (1-\tau_i^{-1}) \cdot \overline{h_i}(t) + \tau_i^{-1} \cdot h_i^{(t)}$$

If the curvature is close to zero for some component, this can drive η^* to infinity. Thus, to avoid numerical instability (to bound the condition number of the approximated Hessian), it is possible to enforce a lower bound $\overline{h_i} \geq \epsilon$. This addition is not necessary in our experiments, due to the presence of an L2-regularization term.

4.4. Slow-start Initialization

To initialize these estimates, we compute the arithmetic averages over a handful ($n_0 = 0.001 \times$ #traindata) of samples before starting to the main algorithm loop. We find that the algorithm works best with a *slow start* heuristic, where the parameter updates are kept small until the exponential averages become sufficiently accurate. This is achieved by overestimating $\overline{v_i}$ and $\bar{l}$) by a factor C. We find that setting $C = d/10$, as a rule of thumb is both robust and near-optimal, because the value of C has only a transient initialization effect on the algorithm. The supplementary material details how we arrived at this, and demonstrates the low sensitivity empirically.

5. Adaptive Learning Rate SGD

The simplest version of the method views each component in isolation. This form of the algorithm will be called "vSGD" (for "variance-based SGD"). In realistic settings with high-dimensional parameter vector, it is not clear a priori whether it is best to have a single, global learning rate (that can be estimated robustly), a set of local, dimension-specific rates, or block-specific learning rates (whose estimation will be less robust). We propose three variants on this spectrum:

vSGD-l uses *local* gradient variance terms and the local diagonal Hessian estimates, leading to $\eta_i^* = (\overline{g_i})^2/(\overline{h_i} \cdot \overline{v_i})$,

vSGD-g uses a *global* gradient variance term and an upper bound on diagonal Hessian terms: $\eta^* = \sum(\overline{g_i})^2/(h^+ \cdot \bar{l})$,

vSGD-b operates like vSGD-g, but being only global across multiple (architecture-specific) *blocks* of parameters, with a different learning rate per block. Similar ideas are adopted in TONGA (Le Roux et al., 2008). In the experiments, the parameters connecting every two layers of the network are regard as a block, with the corresponding bias parameters in separate blocks.

The pseudocode for vSGD-l is given in Algorithm 4.1, the other cases are very similar; all of them have linear complexity in time and space; in fact, the overhead of vSGD is roughly a factor two, which arises from the additional bbrop pass (which could be skipped in all but a fraction of the updates) – this cost is even less critical because it can be trivially parallelized.

6. Non-stationary Behavior

For an intuitive understanding of the effects of the optimal adaptive learning rate method, and the effect of the approximation, we refer to Figures 1 and 2 of the supplementary material, where we illustrate the oscillatory behavior of SGD, and compare the decrease in the loss function and the accompanying change in learning rates on the noisy quadratic loss function (Equation 2).

In realistic on-line learning scenarios, the curvature or noise level in any given dimension changes over time (for example because of the effects of updating other parameters), and thus the learning rates need to *increase* as well as increase. Of course, no fixed learning rate or fixed cooling schedule can achieve this. To illustrate this, we use the same noisy quadratic loss function, but with abrupt changes of the optimum every 300 timesteps.

Figure 2 shows how vSGD with its adaptive memory-size appropriately handles such cases. Its initially large learning rate allows it to quickly approach the optimum, then it gradually reduces the learning rate as the gradient variance increases relative to the squared norm of the average gradient, thus allowing the parameters to closely approach the optimum. When the data distribution changes (abruptly, in our case), the algorithm automatically detects that the norm of the average gradient increased relative to the variance. The learning rate jumps back up and adapts to the new circumstances. Note that the curvature is always 1, which implies that the preconditioning by the diagonal Hessian component vanishes, and still the advantage of adaptive learning rates is clear.

7. Experiments

We test the new algorithm on well-known benchmark problems for digit recognition, image classification and image reconstruction, using the new SGD variants to train both convex models (logistic regression) and non-convex ones (multi-layer perceptrons).

SGD is one of the most common training algorithms in use for (large-scale) neural network training. The experiments in this section compare the three vSGD variants introduced above with SGD, and some adaptive algorithms described in section 2 (AdaGrad, Almeida, Amari and SMD), with additional details in the supplementary material.

We exhaustively search for the best hyper-parameter settings among $\eta_0 \in \{10^{-7}, 3 \cdot 10^{-7}, 10^{-6}, \ldots, 3 \cdot 10^{0}, 10^{1}\}$, $\gamma \in \{0, 1/3, 1/2, 1\}/\#\text{traindata}$, $\tau \in \{10^5, 5 \cdot 10^4, 10^4, 5 \cdot 10^3, 10^3, 10^2, 10^1, \}$ and $\mu \in \{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}\}$ as determined by their lowest test error (averaged over 2 runs), for each individual benchmark setup. The last line in Table 3 shows the total number of settings over which the tuning is done.

7.1. Datasets

We choose two widely used standard datasets to test the different algorithms; the MNIST digit recognition dataset (LeCun & Cortes, 1998) (with 60k train-

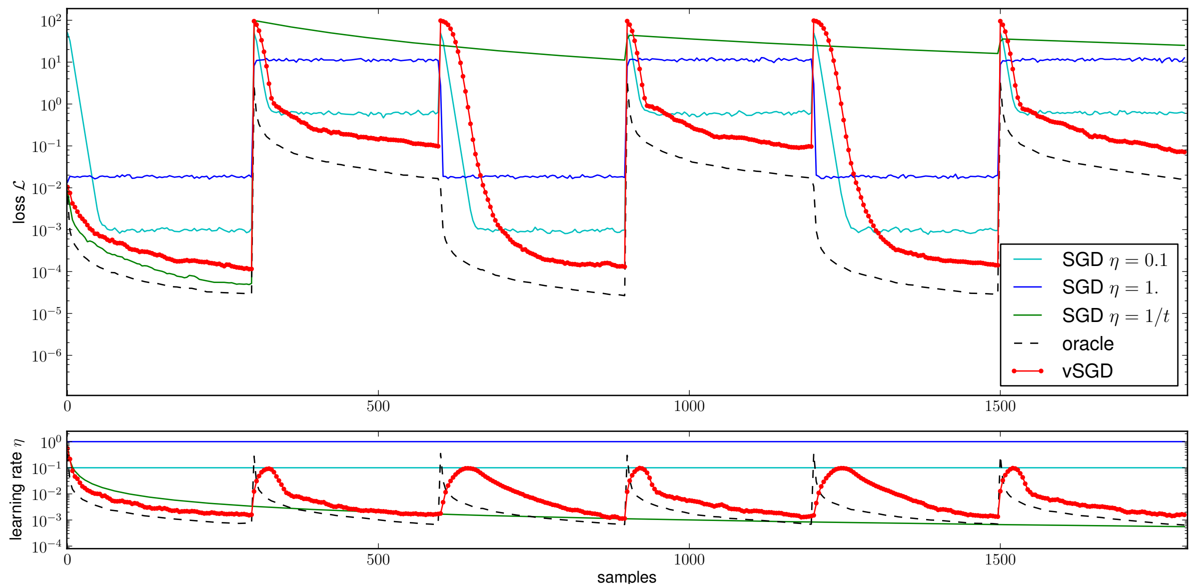

Figure 2. Non-stationary loss. The loss is quadratic but the target value (μ) changes abruptly every 300 time-steps. Above: loss as a function of time, below: corresponding learning rates. This illustrates the limitations of SGD with fixed or decaying learning rates (full lines): any fixed learning rate limits the precision to which the optimum can be approximated (progress stalls); any cooling schedule on the other hand cannot cope with the non-stationarity. In contrast, our adaptive setting ('vSGD', red circles), as closely resembles the optimal behavior (oracle, black dashes). The learning rate decays like $1/t$ during the static part, but increases again after each abrupt change (with just a very small delay compared to the oracle). The average loss across time is substantially better than for any SGD cooling schedule.

	Loss	Network layer sizes	SGD		AdaGrad	Amari		SMD			Almeida	
			η_0	γ	η_0	η_0	τ	η_0	μ	τ	η_0	τ
M0	CE	[784, 10]	$3 \cdot 10^{-2}$	1	10^{-1}	10^{-5}	10^4	10^{-1}	10^{-3}	10^3	10^{-3}	10^3
M1		[784, 120, 10]	$3 \cdot 10^{-2}$	$1/2$	10^{-1}	10^{-6}	$5 \cdot 10^3$	$3 \cdot 10^{-2}$	10^{-4}	10^4	10^{-3}	10^4
M2		[784, 500, 300, 10]	10^{-2}	$1/2$	$3 \cdot 10^{-2}$	$3 \cdot 10^{-7}$	$5 \cdot 10^3$	$3 \cdot 10^{-2}$	10^{-3}	10^2	10^{-3}	10^4
C0	CE	[3072, 10]	$3 \cdot 10^{-3}$	1	10^{-2}							
C1		[3072, 360, 10]	10^{-2}	1	$3 \cdot 10^{-3}$							
CR	MSE	[3072, 120, 3072]	$3 \cdot 10^{-3}$	1	10^{-2}							

Table 1. Experimental setup for standard datasets MNIST and and the subset of CIFAR-10 using neural nets with 0 hidden layer (M0 and C0), 1 hidden layer (M1, C1 and CR), 2 hidden layers (M2). Columns 4 through 13 give the best found hyper-parameters for SGD and the four adaptive algorithms used to compare vSGD to. Note that those hyper-parameters vary substantially across the benchmark tasks.

ing samples, and 10k test samples), and the CIFAR-10 small natural image dataset (Krizhevsky, 2009), namely the 'batch1' subset, which contains 10k training samples and 10k test samples. We use CIFAR both to learn image classification and reconstruction. The only form of preprocessing used (on both datasets) is to normalize the data by substracting mean of the training data along each input dimension.

7.2. Network Architectures

We use four different architectures of feed-forward neural networks. The first one is simple softmax regression (i.e., a network with no hidden layer) for multi-class classification. It has *convex* loss (cross-entropy) relative to parameters. This setup is denoted 'M0' for the MNIST case, and 'C0' for the CIFAR classification case. The second one (denoted 'M1'/'C1') is a fully connected multi-layer perceptron, with a single hidden layers, with tanh non-linearities at the hidden units. The cross-entropy loss function is non-convex. The third (denoted 'M2', only used on MNIST) is a deep, fully connected multi-layer perceptron, with a two hidden layers, again with tanh non-linearities. The fourth architecture is a simple autoencoder (denoted 'CR'), with one hidden layer (tanh non-linearity) and non-coupled reconstruction weights. This is trained to minimize the mean squared reconstruction error.

	vSGD-l	vSGD-b	vSGD-g	SGD	AdaGrad	SMD	Amari	Almeida
M0	**6.72%**	7.63%	8.20%	7.05%	6.97%	7.02%	7.33%	11.80%
M1	**0.18%**	0.78%	3.50%	0.30%	0.58%	0.40%	2.91%	8.49%
M2	**0.05%**	0.33%	2.91%	0.46%	0.41%	0.55%	1.68%	7.16%
C0	**45.61%**	52.45%	56.16%	54.78%	54.36%	–	–	–
C1	**33.16%**	45.14%	54.91%	47.12%	45.20%	–	–	–
CR	10.64	10.13	15.37	**9.77**	**9.80**	–	–	–

Table 2. Final classification error (and reconstruction error for CIFAR-2R) on the **training** set, obtained after 6 epochs of training, and averaged over ten random initializations. Variants are marked in bold if they don't differ statistically significantly from the best one ($p = 0.01$). Note that the tuning parameters of SGD, AdaGrad, SMD, Amari and Almeida are different for each benchmark (see Table 1). We observe the best results with the full element-wise learning rate adaptation ('vSGD-l'), almost always significantly better than the best-tuned SGD or best-tuned AdaGrad.

	vSGD-l	vSGD-b	vSGD-g	SGD	AdaGrad	SMD	Amari	Almeida
M0	**7.50%**	7.89%	8.20%	**7.60%**	**7.52%**	**7.57%**	**7.69%**	11.13%
M1	**2.42%**	**2.44%**	4.14%	**2.34%**	2.70%	**2.37%**	3.95%	8.39%
M2	**2.16%**	**2.05%**	3.65%	**2.15%**	2.34%	**2.18%**	2.97%	7.32%
C0	66.05%	61.70%	**61.10%**	**61.06%**	**61.25%**	–	–	–
C1	**57.72%**	59.55%	60.62%	58.85%	58.67%	–	–	–
CR	11.05	10.57	15.71	**10.29**	**10.33**	–	–	–
#settings	1	1	1	68	17	476	119	119

Table 3. Final classification error (and reconstruction error for CIFAR-2R) on the **test** set, after 6 epochs of training, averaged over ten random initializations. Variants are marked in bold if they don't differ statistically significantly from the best one ($p = 0.01$). Note that the parameters of SGD, AdaGrad, SMD, Amari and Almeida were finely tuned, on this same test set, and were found to be different for each benchmark (see Table 1); the last line gives the total number of parameter settings over which the tuning was performed. Compared to training error, test set performance is more balanced, with vSGD-l being better or statistically equivalent to the best-tuned SGD in 4 out of 6 cases. The main outlier (C0) is a case where the more aggressive element-wise learning rates led to overfitting (compare training error in Table 2).

Again, the loss is non-convex w.r.t. the parameters.

Formally, given input data $h_0 = x$, the network processes sequentially through $H > 0$ hidden layers by applying affine transform then an element-wise tanh,

$$h_{k+1} = \tanh(W_k h_k + b_k), \quad k = 0, \cdots, H-1.$$

The output of the network $y = h_{H+1} = W_H h_H + b_H$ is then feed into the loss function. For cross-entropy loss, the true label c gives the target (delta) distribution to approximate, thus the loss is

$$\mathbb{E}[KL(\delta_c || p_y)] = \mathbb{E}[-\log(p_y(c))],$$

where

$$p_y(c) = \frac{\exp^{-y(c)}}{\sum_k \exp^{-y(k)}}.$$

For mean-squared reconstruction error, the loss is $\mathbb{E}[\frac{1}{2}||x-y||_2^2]$ The exact numbers of hidden units in each layer, and the corresponding total problem dimensions are given in Table 1. The parameters are initialized randomly based on Glorot & Bengio.

To avoid over-fitting, especially for CIFAR which has a comparatively small dataset, we add $\frac{\lambda}{2}||w||_2^2$, a L^2 regularization term on the weights, to the loss in all experiments (with $\lambda = 10^{-4}$). This also avoids numerical instability in vSGD-l, because the estimated diagonal Hessian elements will almost never be close to zero.

7.3. Results

For each benchmark, ten independent runs are averaged and reported in Table 2 (training set) and Table 3 (test set). They show that the best vSGD variant, across the board, is vSGD-l, which most aggressively adapts one learning rate per dimension. It is almost always significantly better than the best-tuned SGD or best-tuned AdaGrad in the training set, and better or statistically equivalent to the best-tuned SGD in 4 out of 6 cases on the test set. The main outlier (C0) is a case where the more aggressive element-wise learning rates led to overfitting (compare training error in Table 2), probably because of the comparatively small dataset size. Figure 3 illustrates the sensitivity to hyper-parameters of SGD, AdaGrad, SMD and Amari's natural gradient on the three MNIST benchmarks: different settings scatter across the performance scale adn tuning matters. This is in stark

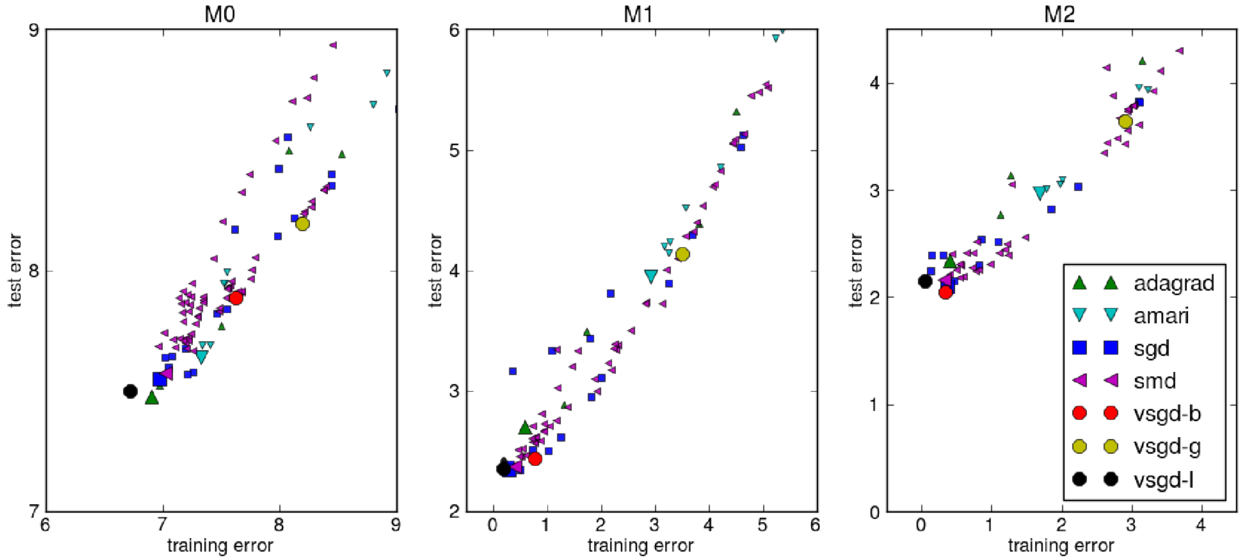

Figure 3. Training error versus test error on the three MNIST setups (after 6 epochs). Different symbol-color combinations correspond to different algorithms, with the best-tuned parameter setting shown as a much larger symbol than the other settings tried (the performance of Almeida is so bad it's off the charts). The axes are zoomed to the regions of interest for clarity, for a more global perspective, and for the corresponding plots on the CIFAR benchmarks, see the supplementary material. Note that there was no tuning for our parameter-free vSGD, yet its performance is consistently good (see black circles).

contrast with vSGD, which without tuning obtains the same performance than the best-tuned algorithms. Numerous additional visualization, e.g., the evolution of learning rates, are provided in the supplementary material.

8. Conclusions

Starting from the idealized case of quadratic loss contributions from each sample, we derived a method to compute an optimal learning rate at each update, and (possibly) for each parameter, that optimizes the expected loss after the next update. The method relies on the square norm of the expectation of the gradient, and the expectation of the square norm of the gradient. A proof of the convergence in probability is given. We showed different ways of approximating those learning rates in linear time and space in practice. The experimental results confirm the theoretical prediction: the adaptive learning rate method completely eliminates the need for manual tuning of the learning rate, or for systematic search of its best value.

Our adaptive approach makes SGD more robust in two related ways: (a) When used in on-line training scenarios with non-stationary signals, the adaptive learning rate automatically increases when the distribution changes, so as to adjust the model to the new distribution, and automatically decreases in stable periods when the system fine-tunes itself within an attractor. This provides robustness to dynamic changes of the optimization landscape. (b) The tuning-free property implies that the same algorithm can adapt to drastically different circumstances, which can appear within a single (deep or heterogeneous) network. This robustness alleviates the need for careful normalizations of inputs and structural components.

Given the successful validation on a variety of classical large-scale learning problems, we hope that this enables for SGD to be a truly user-friendly 'out-of-the-box' method.

Future work will address how our approach can be extended to distinctly non-separable loss functions (e.g., where the diagonal of the Hessian is zero). Some complementary results are already available, and presented in a companion paper (Schaul & LeCun, 2013), discussing minibatch parallelization, reweighted updates for sparse gradients, and robustness on non-smooth loss functions.

Acknowledgments

The authors want to thank Camille Couprie, Clément Farabet and Arthur Szlam for helpful discussions, and Shane Legg for the paper title. This work was funded in part through AFR postdoc grant number 2915104, of the National Research Fund Luxembourg, and ONR Grant 5-74100-F6510.

References

Almeida, LB and Langlois, T. Parameter adaptation in stochastic optimization. *On-line learning in neural networks*, 1999.

Amari, Shun-ichi, Park, Hyeyoung, and Fukumizu, Kenji. Adaptive Method of Realizing Natural Gradient Learning for Multilayer Perceptrons. *Neural Computation*, 12(6):1399–1409, June 2000. ISSN 0899-7667.

Bach, F. and Moulines, E. Non-asymptotic analysis of stochastic approximation algorithms for machine learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2011.

Bordes, Antoine, Bottou, Léon, and Gallinari, Patrick. SGD-QN: Careful quasi-newton stochastic gradient descent. *Journal of Machine Learning Research*, 10: 1737–1754, July 2009.

Bottou, Léon. Online algorithms and stochastic approximations. In Saad, David (ed.), *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

Bottou, Léon and Bousquet, Olivier. The tradeoffs of large scale learning. In Sra, Suvrit, Nowozin, Sebastian, and Wright, Stephen J. (eds.), *Optimization for Machine Learning*, pp. 351–368. MIT Press, 2011.

Bottou, Léon and LeCun, Yann. Large scale online learning. In Thrun, Sebastian, Saul, Lawrence, and Schölkopf, Bernhard (eds.), *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Chapelle, Olivier and Erhan, Dumitru. Improved preconditioner for hessian free optimization. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Duchi, John C., Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. 2010.

George, Abraham P. and Powell, Warren B. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine Learning*, 65(1):167–198, May 2006. ISSN 0885-6125.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10).*

Krizhevsky, Alex. Learning multiple layers of features from tiny images. Technical report, Department of Computer Science, University of Toronto, 2009.

Le Roux, N., Manzagol, P.A., and Bengio, Y. Topmoumoute online natural gradient algorithm, 2008.

Le Roux, Nicolas and Fitzgibbon, Andrew. A fast natural newton method. In *Proceedings of the 27th International Conference on Machine Learning*. Citeseer, 2010.

LeCun, Y., Bottou, L., Orr, G., and Muller, K. Efficient backprop. In Orr, G. and K., Muller (eds.), *Neural Networks: Tricks of the trade*. Springer, 1998.

LeCun, Yann and Cortes, Corinna. The mnist dataset of handwritten digits. 1998. http://yann.lecun.com/exdb/mnist/.

Martens, J, Sutskever, I, and Swersky, K. Estimating the Hessian by Back-propagating Curvature. *arXiv preprint arXiv:1206.6464*, 2012.

Robbins, H. and Monro, S. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

Schaul, Tom and LeCun, Yann. Adaptive learning rates and parallelization for stochastic, sparse, non-smooth gradients. In *Proceedings of the International Conference on Learning Representations (ICLR)*, Scottsdale, AZ, 2013.

Schraudolph, Nicol N. Local gain adaptation in stochastic gradient descent. In *Artificial Neural Networks, 1999. ICANN 99. Ninth International Conference on (Conf. Publ. No. 470)*, volume 2, pp. 569–574. IET, 1999.

Schraudolph, Nicol N. Fast curvature matrix-vector products for second-order gradient descent. *Neural computation*, 14(7):1723–1738, 2002.

Xu, Wei. Towards optimal one pass large scale learning with averaged stochastic gradient descent. *ArXiv-CoRR*, abs/1107.2490, 2011.

No More Pesky Learning Rates: Supplementary Material

Tom Schaul SCHAUL@CIMS.NYU.EDU
Sixin Zhang ZSX@CIMS.NYU.EDU
Yann LeCun YANN@CIMS.NYU.EDU
Courant Institute of Mathematical Sciences
New York University
715 Broadway, New York, NY 10003, USA

A. Convergence Proof

If we do gradient descent with $\eta^*(t)$, then almost surely, the algorithm converges (for the quadratic model). To prove that, we follow classical techniques based on Lyapunov stability theory (Bucy, 1965). Notice that the expected loss follows

$$\begin{aligned}
& \mathbb{E}\left[J\left(\theta^{(t+1)}\right) \mid \theta^{(t)}\right] \\
= \;& \frac{1}{2}h \cdot \mathbb{E}\left[\left((1-\eta^* h)(\theta^{(t)}-\theta^*)+\eta^* h\sigma\xi\right)^2+\sigma^2\right] \\
= \;& \frac{1}{2}h\left[(1-\eta^* h)^2(\theta^{(t)}-\theta^*)^2+(\eta^*)^2h^2\sigma^2+\sigma^2\right] \\
= \;& \frac{1}{2}h\left[\frac{\sigma^2}{(\theta^{(t)}-\theta^*)^2+\sigma^2}(\theta^{(t)}-\theta^*)^2+\sigma^2\right] \\
\leq \;& J\left(\theta^{(t)}\right)
\end{aligned}$$

Thus $J(\theta^{(t)})$ is a positive super-martingale, indicating that almost surely $J(\theta^{(t)}) \to J^\infty$. We are to prove that almost surely $J^\infty = J(\theta^*) = \frac{1}{2}h\sigma^2$. Observe that

$$\begin{aligned}
J(\theta^{(t)}) - \mathbb{E}[J(\theta^{(t+1)}) \mid \theta^{(t)}] &= \frac{1}{2}h\eta^*(t)\,, \\
\mathbb{E}[J(\theta^{(t)})] - \mathbb{E}[J(\theta^{(t+1)}) \mid \theta^{(t)}] &= \frac{1}{2}h\mathbb{E}[\eta^*(t)]
\end{aligned}$$

Since $\mathbb{E}[J(\theta^{(t)})]$ is bounded below by 0, the telescoping sum gives us $\mathbb{E}[\eta^*(t)] \to 0$, which in turn implies that in probability $\eta^*(t) \to 0$. We can rewrite this as

$$\eta^*(t) = \frac{J(\theta_t) - \frac{1}{2}h\sigma^2}{J(\theta_t)} \to 0$$

By uniqueness of the limit, almost surely, $\frac{J^\infty - \frac{1}{2}h\sigma^2}{J^\infty} = 0$. Given that J is strictly positive everywhere, we conclude that $J^\infty = \frac{1}{2}h\sigma^2$ almost surely, i.e $J(\theta^{(t)}) \to \frac{1}{2}h\sigma^2 = J(\theta^*)$.

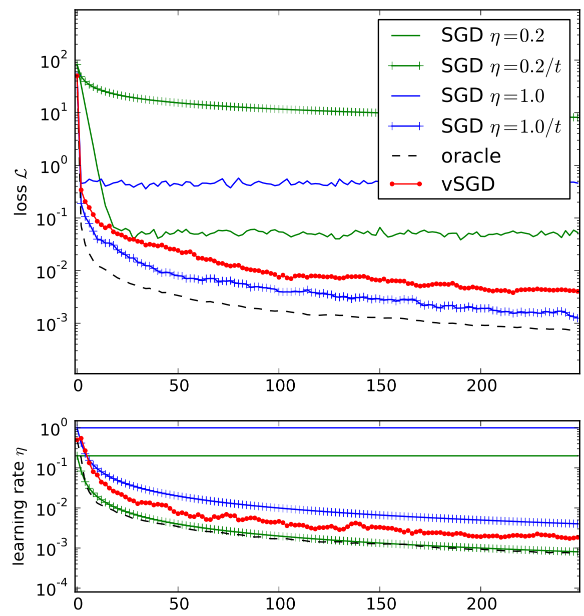

Figure 1. Optimizing a noisy quadratic loss (dimension $d = 1$, curvature $h = 1$). Comparison between SGD for two different fixed learning rates 1.0 and 0.2, and two cooling schedules $\eta = 1/t$ and $\eta = 0.2/t$, and vSGD (red circles). In dashed black, the 'oracle' computes the true optimal learning rate rather than approximating it. In the top subplot, we show the median loss from 1000 simulated runs, and below are corresponding learning rates. We observe that vSGD initially descends as fast as the SGD with the largest fixed learning rate, but then quickly reduces the learning rate which dampens the oscillations and permits a continual reduction in loss, beyond what any fixed learning rate could achieve. The best cooling schedule ($\eta = 1/t$) outperforms vSGD, but when the schedule is not well tuned ($\eta = 0.2/t$), the effect on the loss is catastrophic, even though the produced learning rates are very close to the oracle's (see the overlapping green crosses and the dashed black line at the bottom).

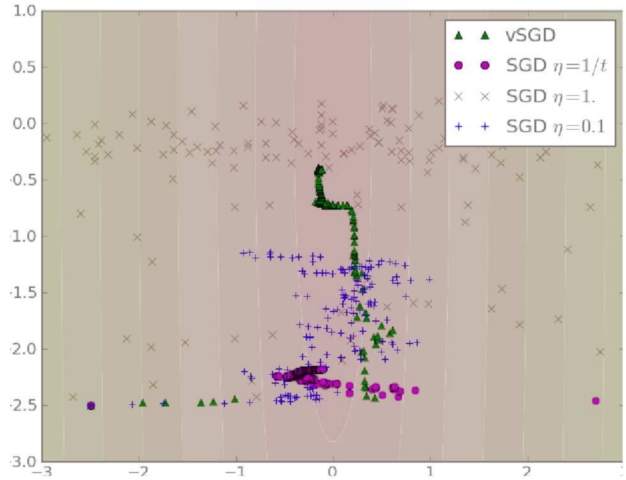

Figure 2. Illustration of the dynamics in a noisy quadratic bowl (with 10 times larger curvature in one dimension than the other). Trajectories of 400 steps from vSGD, and from SGD with three different learning rate schedules. SGD with fixed learning rate (crosses) descends until a certain depth (that depends on η) and then oscillates. SGD with a $1/t$ cooling schedule (pink circles) converges prematurely. On the other hand, vSGD (green triangles) is much less disrupted by the noise and continually approaches the optimum.

B. Derivation of Global Learning Rate

We can derive an optimal global learning rate η_g^* as follows.

$$\begin{aligned}\eta_g^*(t) &= \arg\min_\eta \mathbb{E}\left[\mathcal{J}\left(\boldsymbol{\theta}^{(t+1)}\right) \mid \boldsymbol{\theta}^{(t)}\right] \\ &= \arg\min_\eta \sum_{i=1}^d h_i \Big((1-\eta h_i)^2(\theta_i^{(t)}-\theta_i^*)^2 \\ &\qquad\qquad + \sigma_i^2 + \eta^2 h_i^2 \sigma_i^2\Big) \\ &= \arg\min_\eta \left[\eta^2 \sum_{i=1}^d \left(h_i^3(\theta_i^{(t)}-\theta_i^*)^2 + h_i^3\sigma_i^2\right)\right. \\ &\qquad\qquad \left. -2\eta \sum_{i=1}^d h_i^2(\theta_i^{(t)}-\theta_i^*)^2\right]\end{aligned}$$

which gives

$$\eta_g^*(t) = \frac{\sum_{i=1}^d h_i^2(\theta_i^{(t)}-\theta_i^*)^2}{\sum_{i=1}^d \left(h_i^3(\theta_i^{(t)}-\theta_i^*)^2 + h_i^3\sigma_i^2\right)}$$

The adaptive time-constant for the global case is:

$$\tau_g(t+1) = \left(1 - \frac{\sum_{i=1}^d \overline{g_i}^2}{\overline{l}(t)}\right) \cdot \tau_g(t) + 1$$

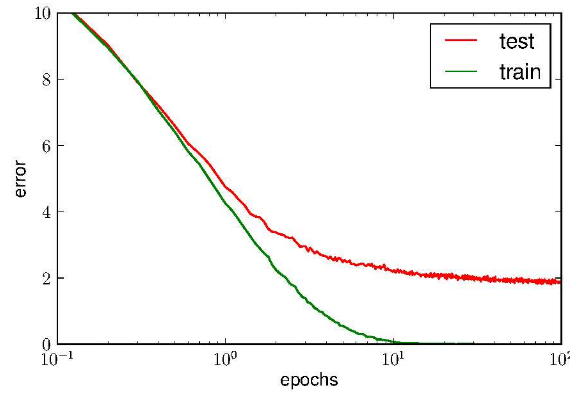

Figure 3. Learning curves for full-length runs of 100 episodes, using vSGD-l on the M1 benchmark with 800 hidden units. Test error is shown in red, training error is green. Note the logarithmic scale of the horizontal axis. The average test error after 100 epochs is 1.87%.

C. SMD Implementation

The details of our implementation of SMD (based on a global learning rates) are given by the following updates:

$$\begin{aligned}\boldsymbol{\theta}_{t+1} &\leftarrow \boldsymbol{\theta}_t - \eta_t \nabla_{\boldsymbol{\theta}} \\ \eta_{t+1} &\leftarrow \eta_t \exp\left(-\mu \nabla_{\boldsymbol{\theta}}^\top \mathbf{v}_t\right) \\ \mathbf{v}_{t+1} &\leftarrow (1-\tau^{-1})\mathbf{v}_t - \eta_t\left(\nabla_{\boldsymbol{\theta}} + (1-\tau^{-1}) \cdot \mathbf{H}_t \mathbf{v}_t\right)\end{aligned}$$

where $\mathbf{Hv}$ denotes the Hessian-vector product with vector $\mathbf{v}$, which can be computed in linear time. The three hyper-parameters used are the initial learning rate η_0, the meta-learning rate μ, and a time constant τ for updating the auxiliary vector $\mathbf{v}$.

D. Additional Results and Analysis

Figures 1 through 8 complement the results from the main paper. First, for an intuitive understanding of the effects of the optimal adaptive learning rate method, and the effect of the approximation, we illustrate the oscillatory behavior of SGD (Figure 2), and compare the decrease in the loss function and the accompanying change in learning rates on the noisy quadratic loss function (Equation 2 of the main paper): Figure 1 shows learning curves, contrasting the effect of fixed rates or fixed schedules to adaptive learning rates, whether in approximation or using the oracle. Complementing the MNIST experiments in the main paper, Figure 3 shows the learning curve on the M1 setup over 100 epochs, much longer than the remainder of the experiments. Further, Figure 4 visualizes test and training error for different algorithm settings on the three CIFAR benchmarks, and Figure 5 visualizes test and training error for all algorithms and their settings on all benchmarks (zoomed out from the region of interest to give different perspective). Figure 6 shows the evolution of (minimal/maximal) learning rates over time, emphasizing the effects of slow-start initialization in our approach. Figure 7 and 8 show the effect of tuning for all benchmarks and a subset of the algorithms.

E. Sensitivity to Initialization

Figure 10 shows that the initialization parameter C does not affect performance, so long as it is sufficiently large. This is not surprising, because its only effect is to slow down the initial step sizes until accurate exponential averages of the interesting quantities can be computed.

There is a *critical* minimum value of C, blow which the algorithm is unstable. Figure 9 shows what those critical values are for 13 different setups with widely varying problem dimension. From these empirical results, we derive our rule-of-thumb choice of $C = d/10$ as a 'safe' pick for the constant (in fact it is even a factor 10 larger than the observed critical value for any of the benchmarks, just to be extra careful).

References

Bucy, R. S. Stability and positive supermartingales. *Journal of Differential Equations*, 1(2):151–155, 1965.

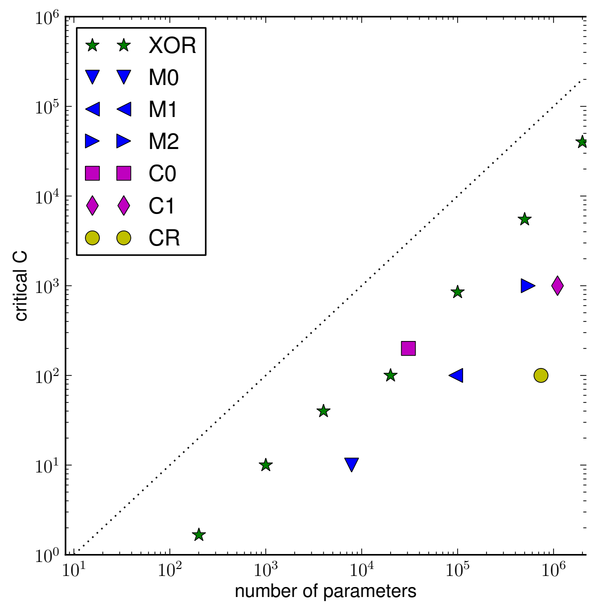

Figure 9. Critical values for initialization parameter C. This plot shows the values of C below which vSGD-l becomes unstable (too large initial steps). We determine the critical C value as the largest for which at least 10% of the runs give rise to instability. The markers correspond to experiments with setups on a broad range of parameter dimensions. Six markers correspond to the benchmark setups from the main paper, and the green stars correspond to simple the XOR-classification task with an MLP of a single hidden layer, the size of which is varied from 2 to 500000 neurons. The black dotted diagonal line indicates, our 'safe' heuristic choice of $C = d/10$.

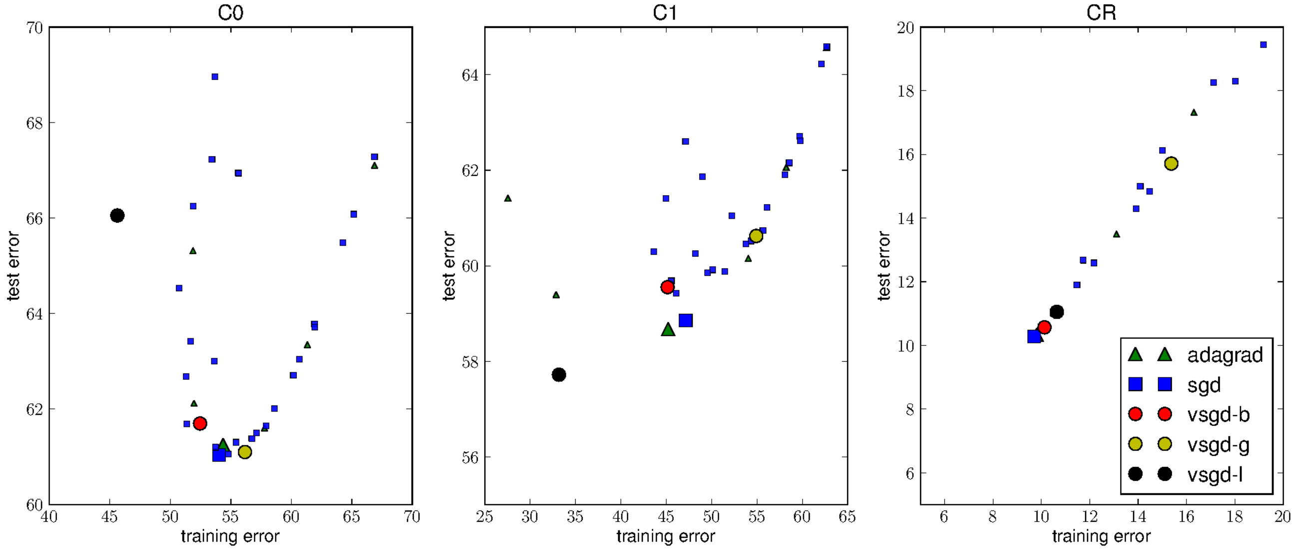

Figure 4. Training error versus test error on the three CIFAR setups (after 6 epochs). Different symbol-color combinations correspond to different algorithms, with the best-tuned parameter setting shown as a much larger symbol than the other settings tried. The axes are zoomed to the regions of interest for clarity. Note how there is much more overfitting here than for MNIST, even with vanilla SGD.

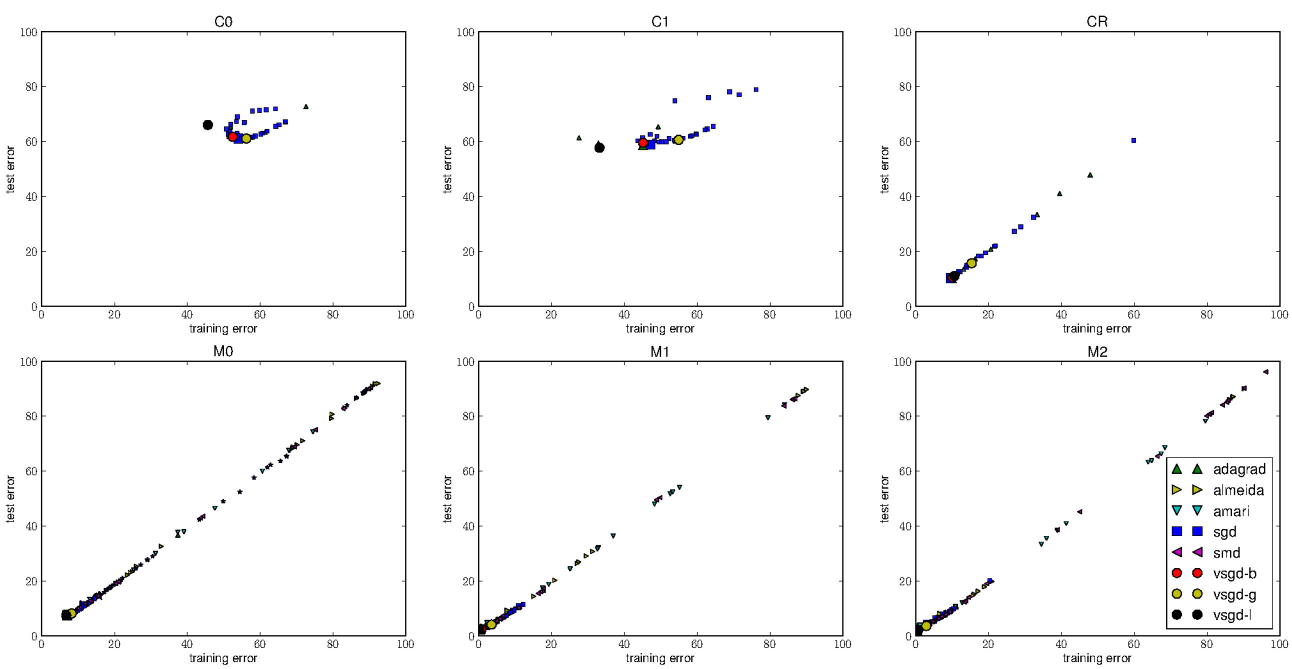

Figure 5. Training error versus test error on all 6 setups, global perspective. Different symbol-color combinations correspond to different algorithms, with the best-tuned parameter setting shown as a much larger symbol than the other settings tried.

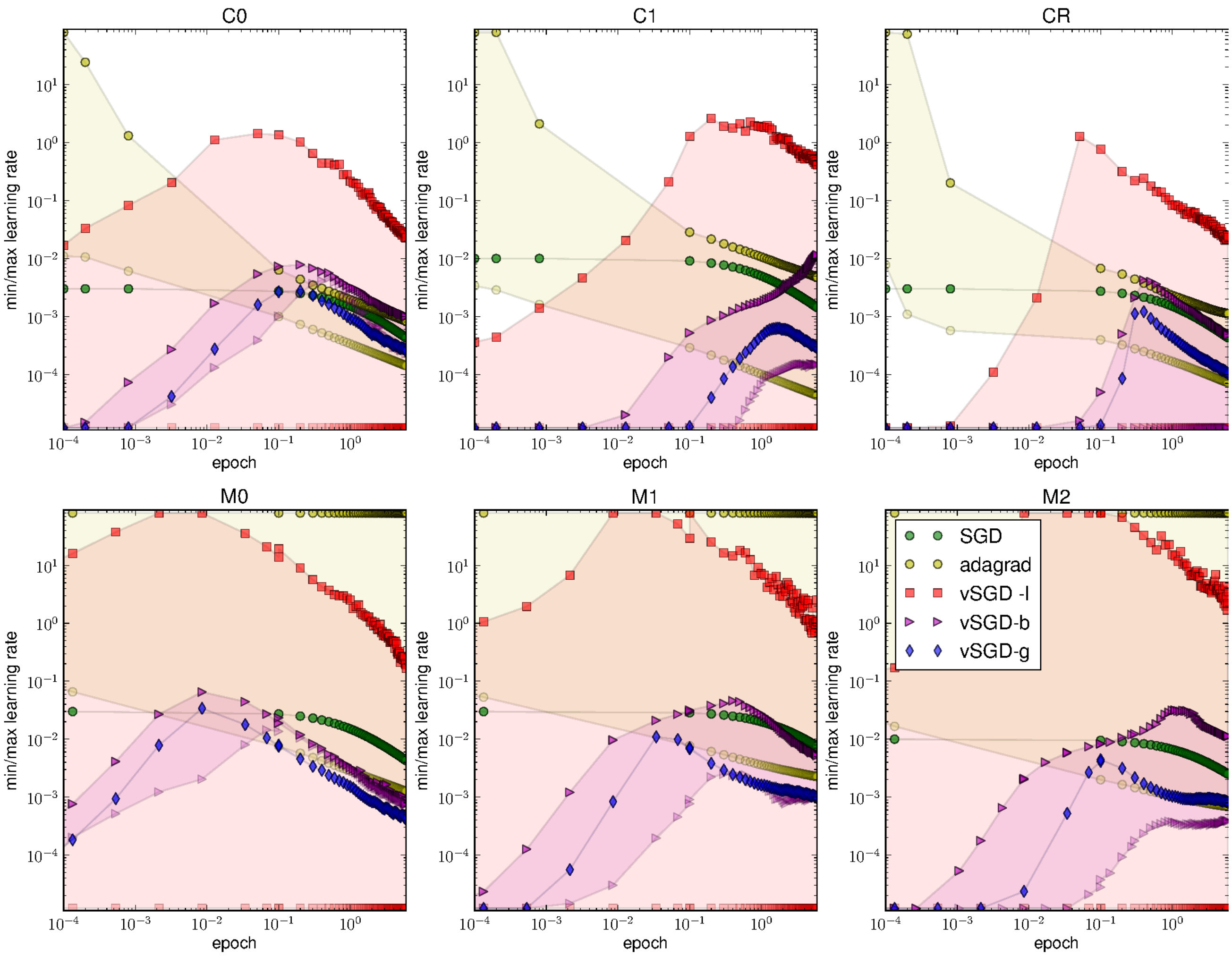

Figure 6. Evolution of learning rates. It shows how the learning rates (minimum and maximum across all dimensions) vary as a function of the epoch. Left: CIFAR classification (no hidden layer), right: MNIST classification (no hidden layer). Each symbol/color corresponds to the median behavior of one algorithm. The range of learning rates (for those algorithms that don't have a single global learning rate) is shown in a colored band in-between the min/max markers. The log-log plot highlights the initial behavior, namely the 'slow start' (until about 0.1 epochs) due to a large C constant in out methods, which contrasts with the quick start of AdaGrad . We also note that AdaGrad (yellow circles) has drastically different ranges of learning rates on the two benchmarks.

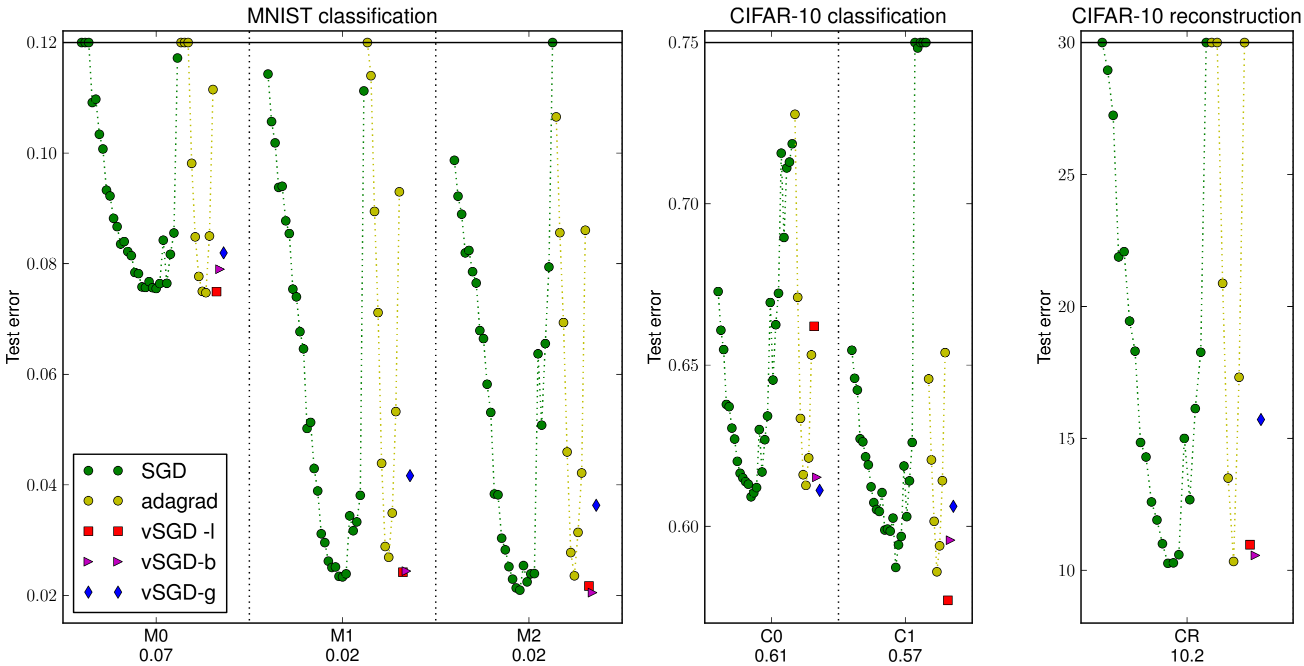

Figure 7. Final performance on test set, on all the practical learning problems (after six epochs) using architectures trained using SGD, AdaGrad or vSGD. We show all 28 SGD settings (green circles), and 11 AdaGrad settings (yellow circles), in contrast to tables 2 and 3, where we only compare the best SGD with the vSGD variants. See also Figure 8 for the corresponding performance on the training set. SGD runs (green circles) vary in terms of different learning rate schedules, and the vSGD variants correspond to the local-global approximation choices described in section 5. Symbols touching the top boundary indicate runs that either diverged, or converged too slow to fit on the scale. Note how SGD tuning is sensitive, and the adaptive learning rates are typically competitive with the best-tuned SGD, and sometimes better.

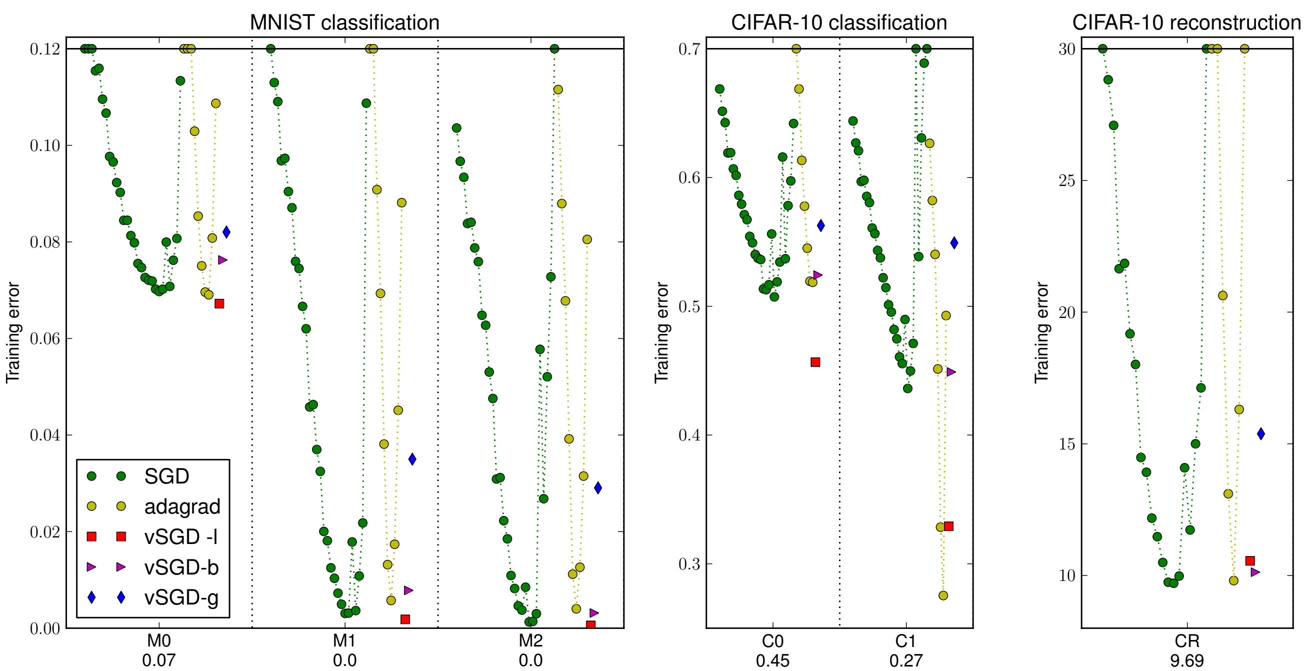

Figure 8. Final performance on training set, on all the practical learning problems (after six epochs) using architectures trained using SGD, AdaGrad or vSGD. We show all 28 SGD settings (green circles), and 11 adaSGD settings (yellow circles). We can observe some clear overfitting in the M2, M3 and C2 cases.

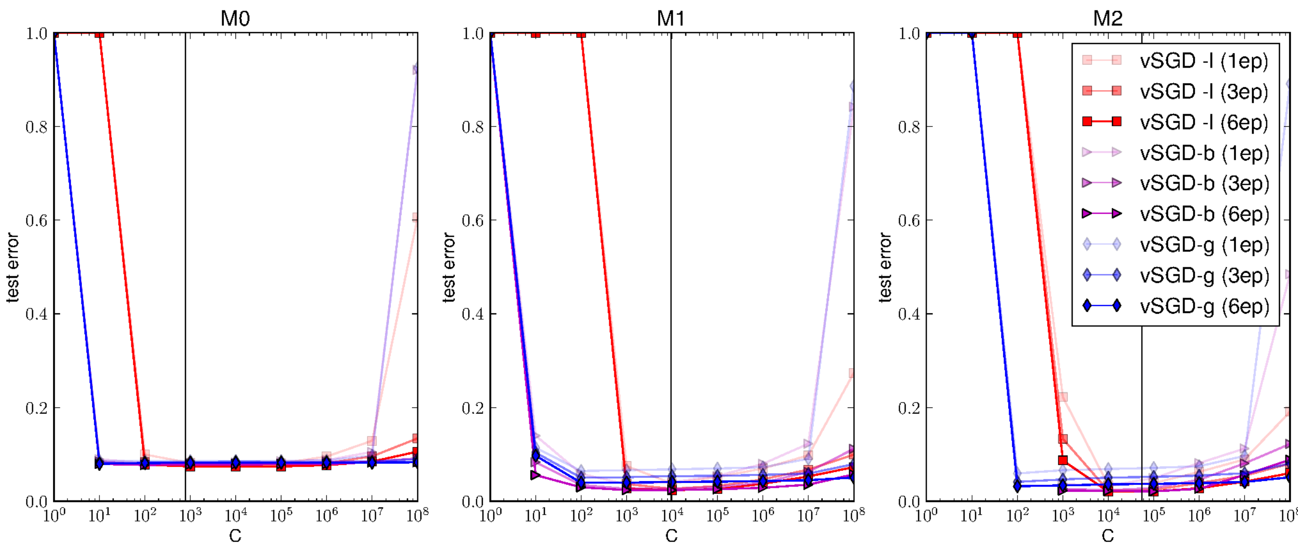

Figure 10. Parameter study on hyper-parameter C. These plots demonstrate that the algorithm is insensitive to the choice of initial slowness parameter C. For each of the setups on the MNIST classification benchmark (with vastly differing parameter dimension d, see Table 1 in the main paper, we show the sensitivity of the test set performance as we vary C over 8 orders of magnitude. Each plot shows the test errors after 1, 3 and 6 epochs (different levels of transparency), for the three adaptive variants (l, b, g, in different colors). In all cases, we find that the updates are unstable if C is chosen too small (the element-wise 'l' variant being most affected), but otherwise C has very little effect, up until when it becomes extremely large: for $C = 10^8$, this initialization basically neutralizes the whole first epoch, and is still felt at epoch 6. The black vertical line indicates, for the three setups, our 'safe' heuristic choice of $C = d/10$. Its only purpose is to avoid instability upon initialization, and given that its 'sweet spot' spans many orders of magnitude, it does not need to be tuned more precisely.