

Mathematical
Institute

Optimization algorithms for training DNNs: Adaptive stepsize, momentum, weight decay, and more

Theories of Deep Learning: C6.5,
Lecture / Video 8

Prof. Jared Tanner
Mathematical Institute
University of Oxford

Oxford
Mathematics

Given a loss function $\mathcal{L}(\theta; X, Y)$, gradient descent is given by

$$\theta^{(k+1)} = \theta^{(k)} - \eta \cdot \text{grad}_\theta \mathcal{L}(\theta, X, Y)$$

with η is referred to as the stepsize, or in DL the "learning rate." In DL $\mathcal{L}(\theta; X, Y)$ is the sum of m individual loss functions for m data point: $\mathcal{L}(\theta; X, Y) = m^{-1} \sum_{\mu=1}^{m} l(\theta; x_\mu, y_\mu)$
For $m \gg 1$ gradient descent is computationally too costly and instead one can break appart the m loss functions into "mini-batches" and repeatedly solve

$$\theta^{(k+1)} = \theta^{(k)} - \eta |S_k|^{-1} \text{grad}_\theta \sum_{\mu \in S_k} l(\theta; x_\mu, y_\mu).$$

This is referred to as stochastic gradient descent as typically S_k is chosen in some randomized method, usually as a partition of $[m]$ and a sequence of S_k which cover $[m]$ is referred to as an "epoch."

Momentum and adaptive diagonal scaling

There are many improvements of SGD typically used in practise for deep learning; particularly popular is Polyak momentum:

$$\theta^{(k+1)} = \theta^{(k)} + \beta(\theta^{(k)} - \theta^{(k-1)}) - \alpha \cdot \text{grad}_\theta \mathcal{L}\left(\theta^{(k)}\right)$$

or Nesterov's accelerated gradient:

$$\begin{aligned} \hat{\theta}^k &= \theta^{(k)} + \beta(\theta^{(k)} - \theta^{(k-1)}) \\ \theta^{(k+1)} &= \hat{\theta}^{(k)} - \alpha \cdot \text{grad}_\theta \mathcal{L}\left(\hat{\theta}^{(k)}\right) \end{aligned}$$

These acceleration methods give substantial improvements in the linear convergence rate for convex problems; linear convergence rates are: Normal GD $\frac{\kappa-1}{\kappa+1}$, Polyak $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ and NAG $\sqrt{\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}}}$.

SGD improvement : Adaptive sub-gradients (Duchi et al. 11')

Preconditioning via past gradient information (AdaGrad)

UNIVERSITY OF OXFORD
Mathematical Institute

Preconditioning improves convergence rate of line-search methods is preconditioning. Let $g^{(k)}(\theta^{(k)}) =: \text{grad}_\theta \mathcal{L}(\theta^{(k)})$ be the gradient of the training loss function at iteration k and

$$B_k(i,i) = \left(\sum_{j=1}^{k} \left(g^{(j)}(\theta^{(j)})(i) \right)^2 \right)^{1/2},$$

the diagonal of the square-root of the sum of prior gradient outer-products. Adaptive sub-gradients (AdaGrad) is preconditioned GD via the diagonal matrix B

$$\theta^{(k+1)} = \theta^{(k)} - \eta |S_k|^{-1} (B^{(k)} + \epsilon I)^{-1} \text{grad}_\theta \sum_{\mu \in S_k} l(\theta; x_\mu, y_\mu).$$

$\epsilon I > 0$ added to avoid poor scaling of small values of $B^{(k)}(i,i)$.

`http://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf`

AdaGrad improvements: RMSProp and AdaDelta

Alternative gradient weighting and step-sizes

AdaGrad preconditions with the inverse of
$B_k(i,i) = \left(\sum_{j=1}^{k} \left(g^{(j)}(\theta^{(j)})(i)\right)^2\right)^{1/2}$.
RMSProp (Hinton) gives more weight to the current gradient

$$B_k^{RMS}(i,i) = \gamma B_{k-1}^{RMS}(i,i) + (1-\gamma)\left(g^{(k)}(\theta^{(k)})(i)\right)^2$$

for some $\gamma \in [0,1]$ and updates as

$$\theta^{(k+1)} = \theta^{(k)} - \eta |S_k|^{-1}(B^{(k)} + \epsilon I)^{-1/2} \text{grad}_\theta \sum_{\mu \in S_k} l(\theta; x_\mu, y_\mu).$$

AdaDelta (Zeiler 12') extends AdaGrad using a similar preconditioned as B_k^{RMS}, but also estimates the stepsize using an average difference in $\theta^{(k)} - \theta^{(k-1)}$.
`https://arxiv.org/abs/1212.5701`

Adaptive moment estimation (Adam) (Kingma et al. 15')

SGD with adaptive momentum

UNIVERSITY OF OXFORD
Mathematical Institute

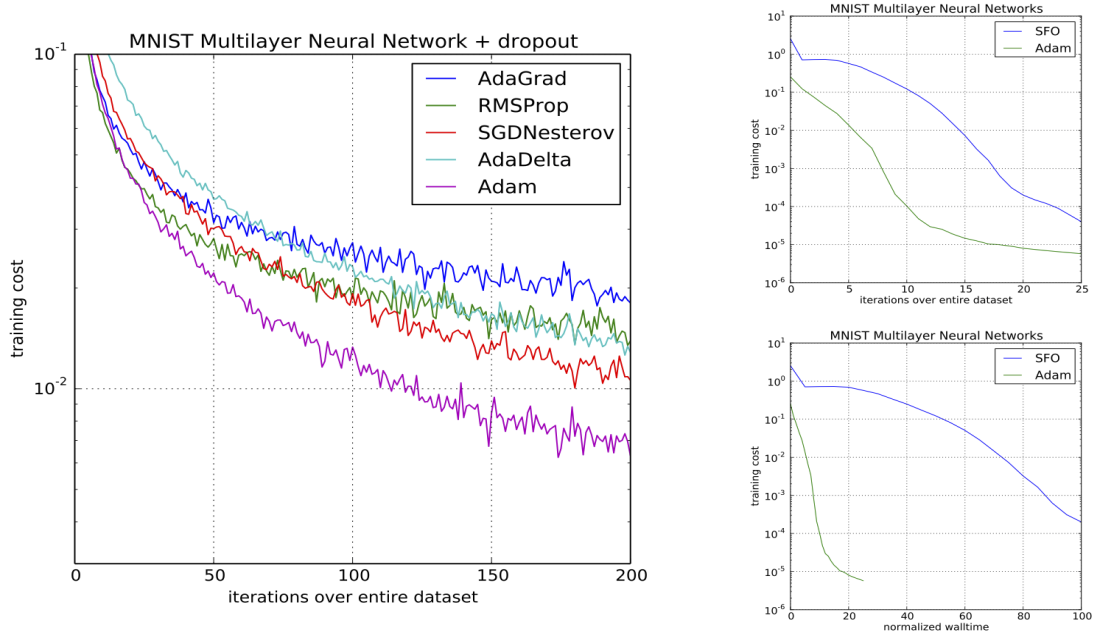
Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)
  $v_0 \leftarrow 0$ (Initialize 2^{nd} moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

https://arxiv.org/pdf/1412.6980.pdf

Adaptive moment estimation (Adam) (Kingma et al. 15')

Training on MNIST



(a) (b)

Figure 2: Training of multilayer neural networks on MNIST images. (a) Neural networks using dropout stochastic regularization. (b) Neural networks with deterministic cost function. We compare with the sum-of-functions (SFO) optimizer (Sohl-Dickstein et al., 2014)

`https://arxiv.org/pdf/1412.6980.pdf`

Adaptive moment estimation (Adam) (Kingma et al. 15')

Training on CNNs for CIFAR-10

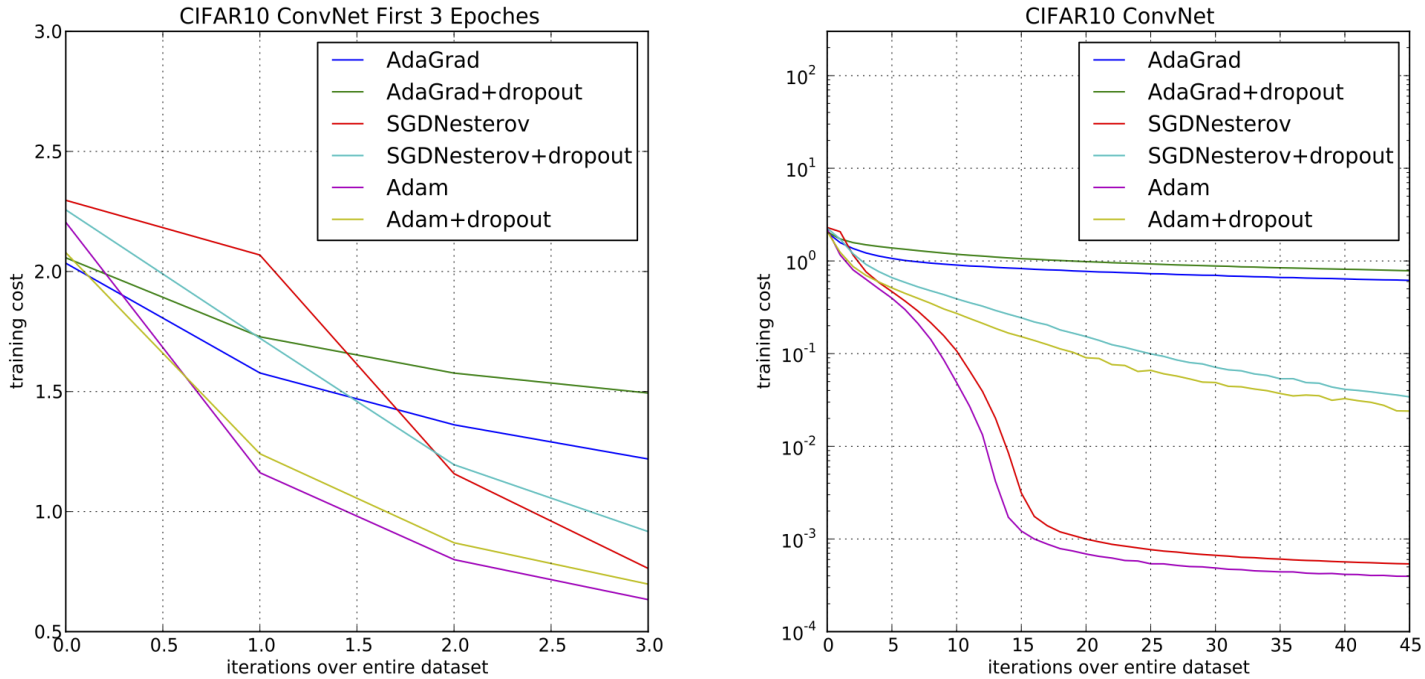


Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

`https://arxiv.org/pdf/1412.6980.pdf`

Scalar adaptive gradient for robustness to initial stepsize

AdaGrad: as adaptive stepsize rule (Ward et al. 18')

Scalar diagonal preconditioning

Mathematical Institute

Let $g^{(k)}(\theta^{(k)}) =: \text{grad}_\theta \mathcal{L}(\theta^{(k)})$ be the gradient of the training loss function at iteration k, AdaGrad preconditions with

$$B_k(i,i) = \left(\sum_{j=1}^{k} \left(g^{(j)}(\theta^{(j)})(i) \right)^2 \right)^{1/2}$$

which is the diagonal of the square-root of the sum of prior gradient outer-products. AdaGrad is the gradient descent method

$$\theta^{(k+1)} = \theta^{(k)} - \eta |S_k|^{-1} (B^{(k)} + \epsilon I)^{-1} \text{grad}_\theta \sum_{\mu \in S_k} l(\theta; x_\mu, y_\mu).$$

A simplified version, focusing on the per iteration (as opposed to per index) update is to let $B_k = b_k I$ where $b_{k+1}^2 = b_k^2 + \|g^{(k)}\|_2^2$. `https://arxiv.org/pdf/1806.01811.pdf`

University of Oxford
Mathematical Institute

Scalar AdaGrad update algorithm: Initialize with $\theta^{(0)}$ and $b_0 > 0$

$$\begin{aligned} b_k^2 &= b_{b-1}^2 + \|\text{grad}_\theta \mathcal{L}(\theta^{(k)})\|_2^2 \\ \theta^{(k)} &= \theta^{(k-1)} - b_k^{-1} \text{grad}_\theta \mathcal{L}(\theta^{(k)}) \end{aligned}$$

For $\mathcal{L}(\theta) \in C_L^1$, that is L minimal for which $\|\text{grad}_\theta \mathcal{L}(\theta_1) - \text{grad}_\theta \mathcal{L}(\theta_2)\|_2 \leq L\|\theta_1 - \theta_2\|$ for all θ_1, θ_2, then scalar batch AdaGrad satisfies $\min_{k=1,\ldots,T-1} \|\text{grad}_\theta \mathcal{L}(\theta^{(k)})\|_2^2 \leq \epsilon$ for either

$$\begin{aligned} T &= 1 + \left\lceil 2\epsilon^{-1} \mathcal{L}(\theta^{(0)})(b_0 + 2\mathcal{L}(\theta^{(0)})) \right\rceil \quad \text{if} \quad b_0 \geq L, \quad \text{or} \\ &= 1 + \left\lceil \epsilon^{-1} \left(L^2 - b_0^2 + 4(\mathcal{L}(\theta^{(0)}) + (3/4 + \log(L/b_0))L)^2 \right) \right\rceil \end{aligned}$$

if $b_0 < L$. In contrast, if b_k is a fixed constant b, then if $b < L/2$ GD can diverge, while if $b \geq L$ then $T = 2b\epsilon^{-1}\mathcal{L}(\theta^{(0)})$.

Scalar AdaGrad: proof ingredients (Ward et al. 18')

Iteration complexity

Mathematical Institute

Iteration complexity for scalar batch AdaGrad following properties for any non-negative values $a_1, \ldots, a_T$ with $a_1 > 0$, (with a_k taking the place of $\|\text{grad}_\theta \mathcal{L}(\theta^{(k)})\|_2^2$)

$$\sum_{\ell=1}^{T} \frac{a_\ell}{\sum_{i=1}^{\ell} a_i} \leq \log\left(\sum_{i=1}^{T} a_i\right) + 1 \quad \text{and} \quad \sum_{\ell=1}^{T} \frac{a_\ell}{\sqrt{\sum_{i=1}^{\ell} a_i}} \leq 2\sqrt{\sum_{i=1}^{T} a_i}.$$

Also, for any fixed $\epsilon \in (0, 1]$ and $L, b_0 > 0$, the iterates $b_{k+1}^2 = b_k^2 + a_k$ has the property that after $N = \lceil \epsilon^{-1}(L^2 - b_o^2) \rceil + 1$ iterations either $\min_{k=0}^{N-1} a_k \leq \epsilon$ or $b_N \geq L$. Lastly, letting k_0 be the first iterate such that $b_{k_0} \geq L$, then for all $k \geq k_0$ $b_k \leq b_{k_0-1} + 2\mathcal{L}(\theta^{(k_0-1)})$ (bounded above) and $\mathcal{L}(\theta^{(k_0-1)}) \leq \frac{L}{2}(1 + 2\log(b_{k_0-1}/b_0))$ (not diverged).

Scalar AdaGrad: stochastic (Ward et al. 18')

Influence of mini-batch or other gradient approximation

Let $g^{(k)}$ be an unbiased estimator of the gradient $\text{grad}_\theta \mathcal{L}(\theta^{(k)})$ of the training loss function at iteration k; that is $\mathbb{E}(g^{(k)}) = \text{grad}_\theta \mathcal{L}(\theta^{(k)})$. Moveover, let there be a uniform bound $\mathbb{E}(\|g^{(k)}\|_2^2) \leq c_g^2$. Then consider the stochastic scalar AdaGrad update as

$$\begin{aligned} b_k^2 &= b_{b-1}^2 + \|g^{(k)}\|_2^2 \\ \theta^{(k)} &= \theta^{(k-1)} - b_k^{-1} g^{(k)}. \end{aligned}$$

Unlike in the batch version of AdaGrad where b_k converges to a fixed stepsize, stochastic AdaGrad converges roughly at the rate $b_k \approx c_g k^{1/2}$. Morevover Ward et al. showed that

$$\min_{\ell=0,\ldots,N-1} \left(\mathbb{E} \|\text{grad}_\theta \mathcal{L}(\theta^{(k)})\|^{4/3} \right)^{3/2} \leq \mathcal{O}\left(\frac{b_0 + c_g}{N} + \frac{c_g}{N^{1/2}} \right) \log(N c_g^2 / b_0^2).$$

Scalar AdaGrad examples (Ward et al. 18')

MNIST with batch gradients

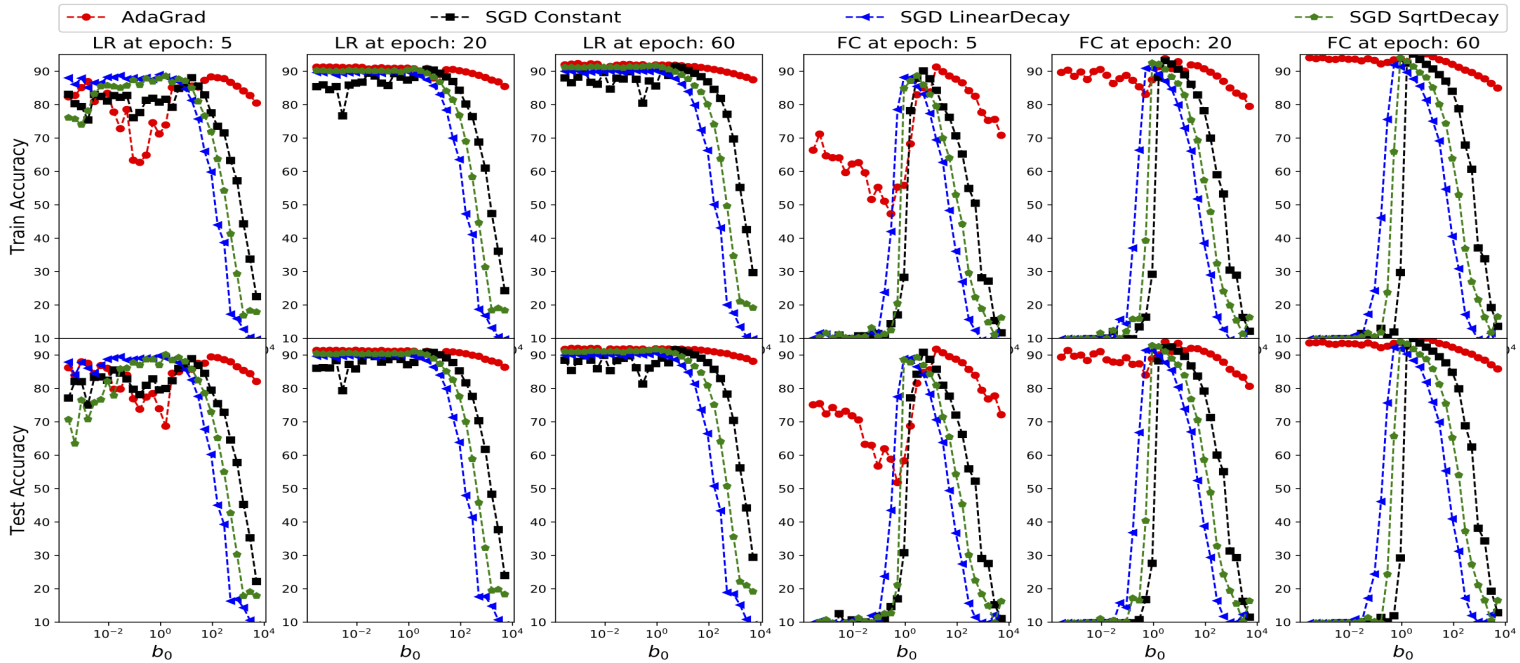


Figure 1: Batch setting on MNIST. Top (bottom) row are plots of train (test) accuracy with respect to the initialization b_0. The left 6 figures are for logistic regression (LR) with snapshots at epoch 5, 20 and 60 in the 1st, 2nd and 3rd column respectively. The right 6 figures are for two fully connected layers (FC) with snapshots at epoch 5, 20 and 60 in the 4th, 5th and 6th column.

https://arxiv.org/pdf/1806.01811.pdf

Scalar AdaGrad examples (Ward et al. 18')

MNIST with mini-batch gradients

University of Oxford
Mathematical Institute

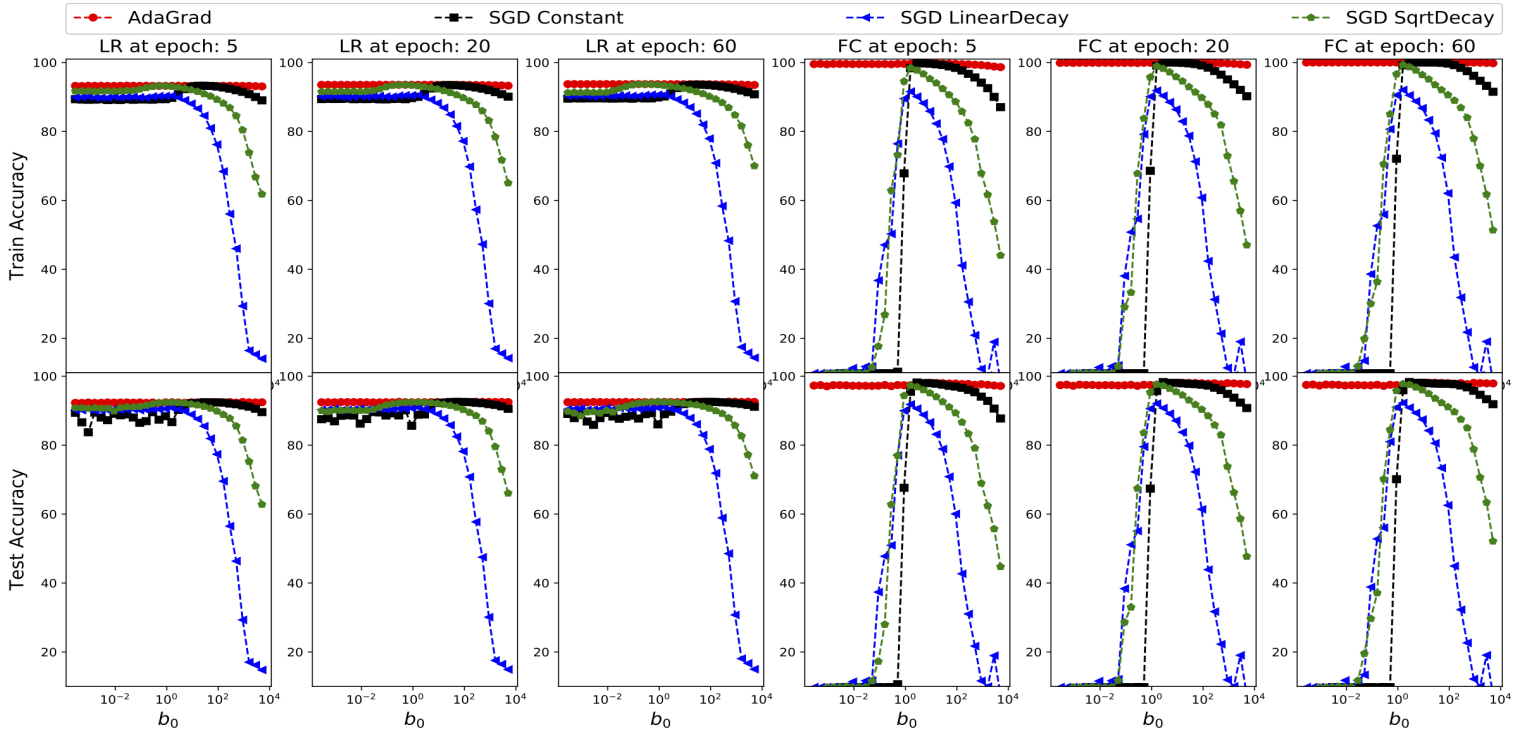


Figure 2: Stochastic setting on MNIST. Left 6 figures by logistic regression and right 6 figures by two fully connected layer. Note that the scale of y-axis change. See Figure 1 for reading instruction.

https://arxiv.org/pdf/1806.01811.pdf

Scalar AdaGrad examples (Ward et al. 18')

CIFAR-10 with mini-batch gradients

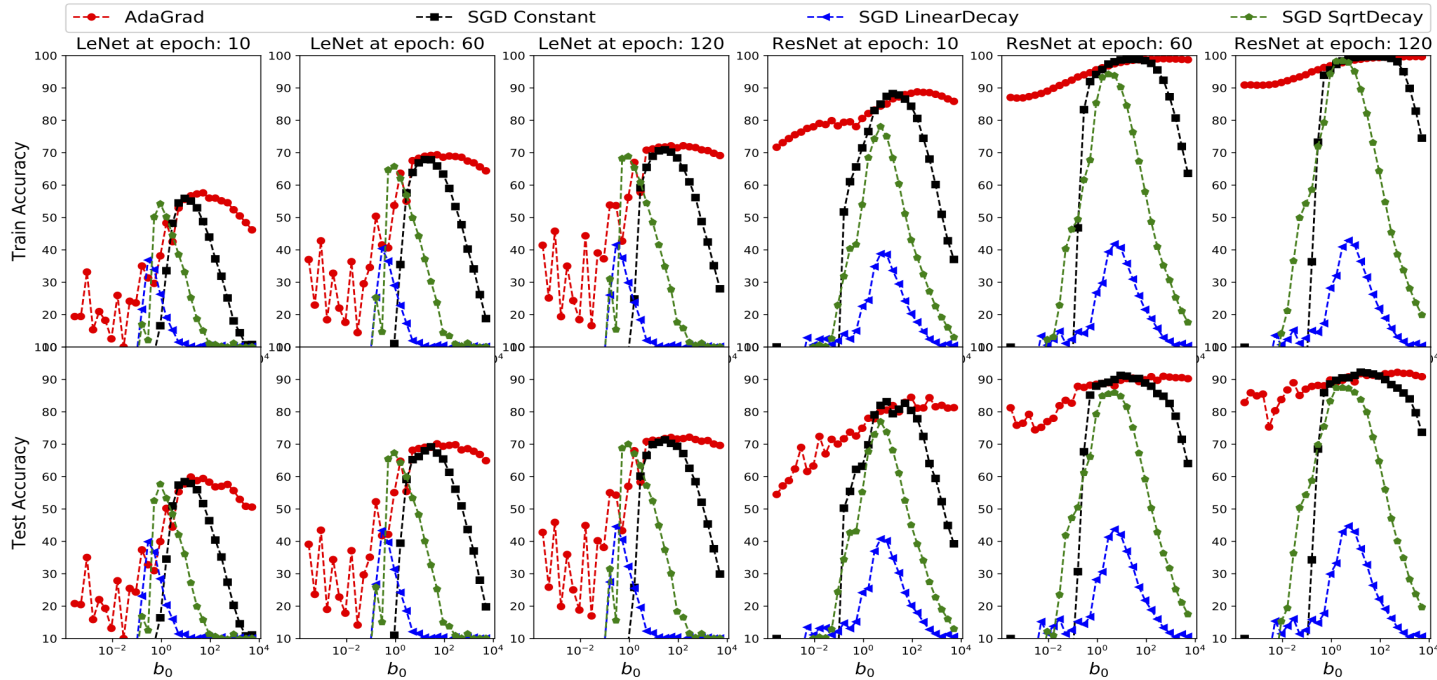


Figure 3: Stochastic setting on CIFAR10. Left 6 figures by LeNet and right 6 figures by ResNet. Note that the epoch (see title) is different from previous figures and no momentum is used. See Figure 1 for reading instruction.

`https://arxiv.org/pdf/1806.01811.pdf`

Weight decay, SGD, Adam, and AdamW

Weight decay balances layers

Weight decay is especially important for ReLU

UNIVERSITY OF OXFORD
Mathematical Institute

- Deep networks benefit from each layer having a similar magnitude of action, this can be seen in part through the conditioning of the loss by layer.
- ReLU allows different constant scaling factors to be moved between layers, e.g. $c^{-1}W_2ReLU(cW_1x)$ is independent of c and has effective $c^{-1}W_2$ and cW_1 weights.
- To overcome this scaling imbalance we typically augment the data fidelity loss $m_{-1}\sum_{k=1}^{m}\|H(x_k;\theta)-y_k\|_2^2$ by adding weight decay $\lambda\|\theta\|_2^2$ for some suitably chosen λ.
- The ℓ^2 norm weight decay encourages weight parameters to generally be of a similar size; note different from ℓ^1 which induces sparsity and ℓ^∞ which encourages equal values such as for quantization.

SGD and Adam differ in weight decay (Loshchilov et al. 19')

Weight decay equivalence for SGD only

SGD minimizing a loss $\tilde{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \frac{\lambda'}{2}\|\theta\|_2^2$ with fixed stepsize α is equivalent to ℓ^2 regularization where the prior weight parameters are dampened by $(1-\lambda)$,

$$\theta^{(k+1)} = (1-\lambda)\theta^{(k)} - \alpha\nabla\mathcal{L}(\theta)_{|\theta(k)},$$

for $\lambda = \alpha\lambda'$, see assignment 3.
For this reason the gradient of $\|\theta\|_2^2$ isn't typically computed, rather the ℓ^2 regularization is used. However, this equivalence isn't true for Adam type methods. This motivates AdamW which implements weight decay correctly for Adam.
`https://arxiv.org/pdf/1711.05101`

AdamW algorithm (Loshchilov et al. 19')

Subtle changes hae substantial impact; used for ChatGPT

UNIVERSITY OF OXFORD
Mathematical Institute

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda \boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda \boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

https://arxiv.org/pdf/1711.05101

AdamW loss function (Loshchilov et al. 19')

Reduced loss function and larger region

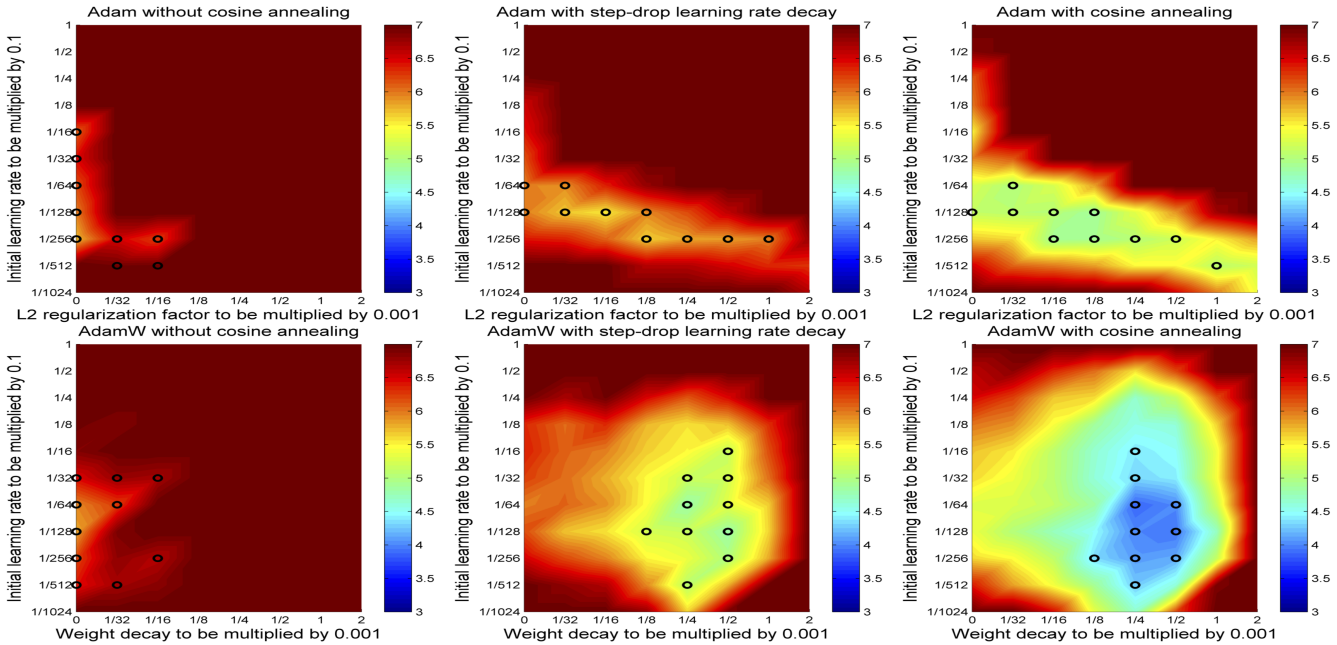


Figure 1: Adam performs better with decoupled weight decay (bottom row, AdamW) than with L_2 regularization (top row, Adam). We show the final test error of a 26 2x64d ResNet on CIFAR-10 after 100 epochs of training with fixed learning rate (left column), step-drop learning rate (with drops at epoch indexes 30, 60 and 80, middle column) and cosine annealing (right column). AdamW leads to a more separable hyperparameter search space, especially when a learning rate schedule, such as step-drop and cosine annealing is applied. Cosine annealing yields clearly superior results.

https://arxiv.org/pdf/1711.05101

AdamW Test error (Loshchilov et al. 19')

Superior test error by epoch and decay rate

University of Oxford
Mathematical Institute

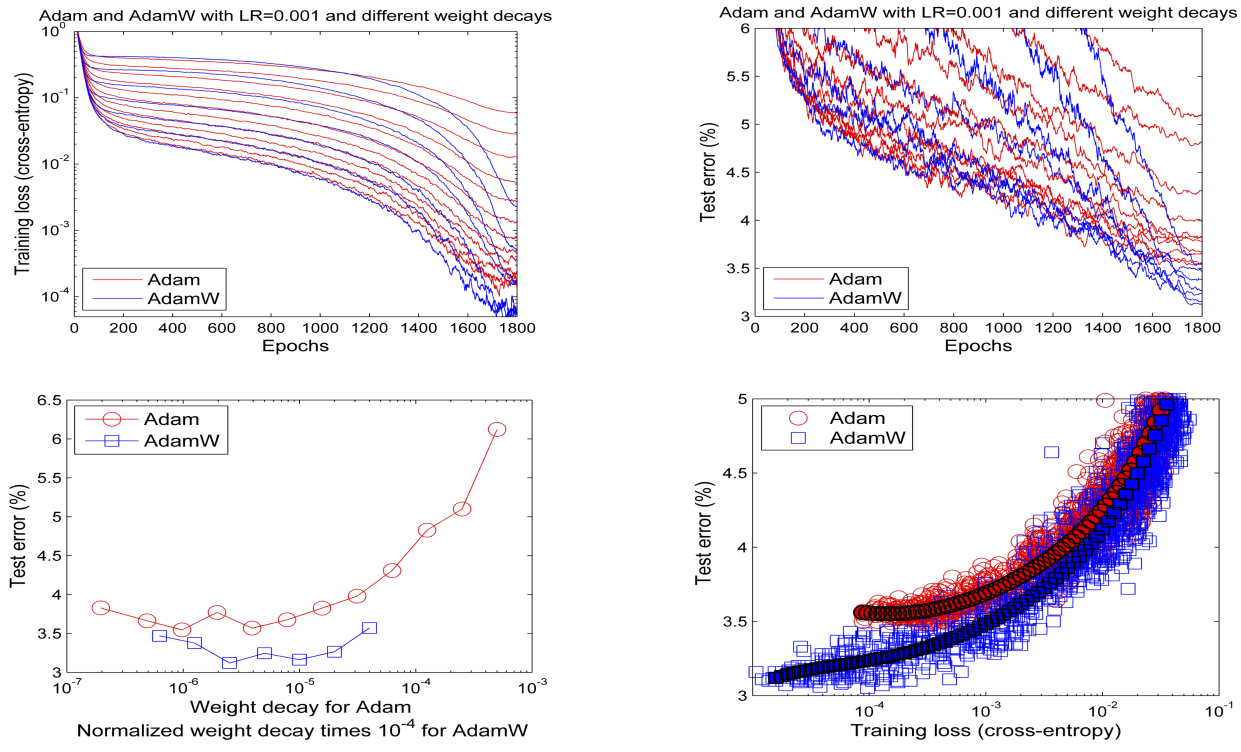


Figure 3: Learning curves (top row) and generalization results (bottom row) obtained by a 26 2x96d ResNet trained with Adam and AdamW on CIFAR-10. See text for details. SuppFigure 4 in the Appendix shows the same qualitative results for ImageNet32x32.

https://arxiv.org/pdf/1711.05101

Other optimization for ML variants

- K-FAC (Martens et al. 15') incorporates second order curvature information and an information theory perspective with Fisher information. `https://arxiv.org/pdf/1503.05671`
- Adafactor (Shazeer et al. 18') reduce the number of gradient step-sizes used in the diagonal scaling to average over rows and/or columns of weight matrices which reduces the memory needed `https://arxiv.org/pdf/1804.04235`
- LARS (You et al. 17') uses different learning rates per layer rather than per entry. `https://arxiv.org/pdf/1708.03888`
- LAMB (You et al. 19') adapts LARS from focus on CNNs to LLMs, BERT in particular. This uses large batch sizes to reduce gradient variance. `https://arxiv.org/pdf/1904.00962`

MomentUm Orthogonalized by Newton–Schulz

Muon: operator norm steepest descent (Bernstein et al. 24')

Neural network weights are operators, not simply vectors

Mathematical Institute

Gradient descent implicitly measures the size of a matrix update using the Frobenius norm as if the weights are a vector in ℓ_2. However, the weight matrices are instead used as operators on inputs in which case the more natural constraint is the operator norm, the dominant singular value.

$$\max_{\|x\|_2=1} \|\Delta W x\|_2 = \|\Delta W\|$$

If $G = U\Sigma V^T$, steepest descent subject to an operator norm constraint instead gives $\Delta W \propto -UV^T$, the gradient singular values are flattened: $\Sigma_{ii} = 1$ for all i. This avoids anisotropic updates of W.

`https://arxiv.org/abs/2409.20325`

Muon optimizer (Jordan et al., 24')

Momentum followed by approximate orthogonalization

Muon (**M**oment**U**m **O**rthogonalized by **N**ewton–Schulz) applies this idea to the momentum gradient:

$$M_t = \mu M_{t-1} + G_t, \qquad M_t = U\Sigma V^T \quad \longrightarrow \quad \widehat{M}_t := UV^T.$$

$$W_{t+1} = W_t - \eta \widehat{M}_t.$$

Rather than computing an expensive SVD, Muon uses a few Newton–Schulz iterations consisting primarily of matrix multiplications to approximate the singular vectors.

`https://kellerjordan.github.io/posts/muon/`

Muon for LLM training (Liu et al., 25')

Can train more efficiently than AdamW

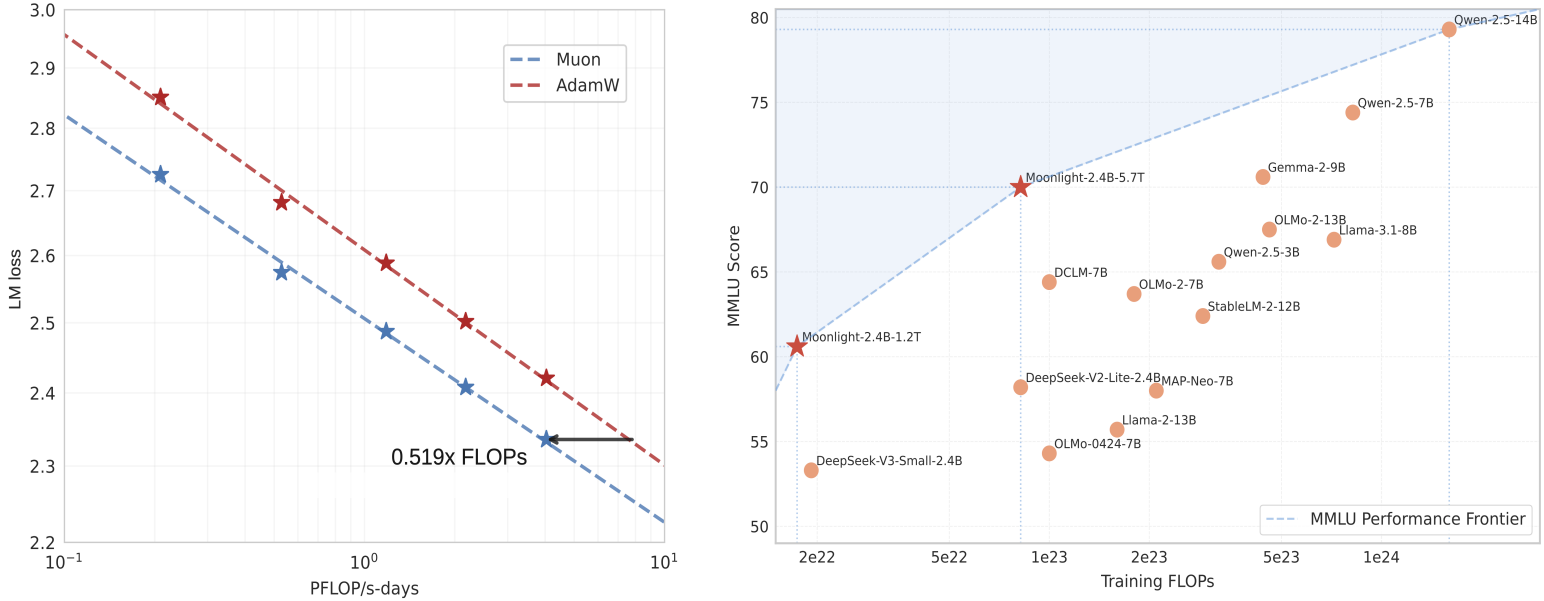


Figure 1: Scaling up with Muon. **(a)** Scaling law experiments comparing Muon and Adam. Muon is $\sim 2\times$ more computational efficient than Adam with compute optimal training. **(b)** The MMLU performance of our Moonlight model optimized with Muon and other comparable models. Moonlight advances the Pareto frontier of performance vs training FLOPs.

`https://arxiv.org/abs/2502.16982`