Outline for today

- Methods for learning the net parameters:
  - Stochastic gradient descent (SGD)
  - Polyak and Nesterov momentum
  - AdaGrad and variants toward Adam
- Simplified version of AdaGrad with scalar, adaptive, stepsize
- Convergence dependence of scalar AdaGrad on initial stepsize

Stochastic gradient descent (SGD)

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

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

where η is referred to as the stepsize, or in deep learning the "learning rate."

Recall, we typically have a loss function which is the sum of n individual loss functions, independent for each data point: $\mathcal{L}(\theta; X, Y) = n^{-1} \sum_{\mu=1}^{n} l(\theta; x_\mu, y_\mu)$

For $n \gg 1$ gradient descent is computationally too costly and instead one can break appart the n loss functions into "mini-batches" and repeatedly solve

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

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

Stochastic gradient descent: challenges and benefits

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

- SGD is preferable for large n as it reduces the per iteration computational cost dependence on n to instead depend on $|\Lambda_k|$ which can be set by the user as opposed to n which is given by the data set.
- SGD, and gradient descent, require selection of a learning rate (stepsize) which in deep learning is typically selected using some costly trial and error heuristics.
- The learning rate is typically chosen adaptively in a way that satisfies $\sum_{k=1}^{\infty}\eta_k = \infty$ and $\sum_{k=1}^{\infty}\eta_k^2 < \infty$; in particular as $\eta_k \sim k^{-1}$.
- The optimal selection of learning weight, and selection of Λ, depends on the unknown local Lipschitz constant $\|\text{grad}l(\theta_1; x_\mu, y_\mu) - \text{grad}l(\theta_2; x_\mu, y_\mu)\| \leq L_\mu\|\theta_1 - \theta_2\|$.

SGD improvements: momentum

There are many improvements of stochastic gradient descent 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 improv. : Adaptive sub-gradients (Duchi et al. 11'[1])

A standard method for improving the convergence rate of line-search methods is preconditioning, Adaptive sub-gradients (AdaGrad) is such a method.
Let $g^{(k)}(\theta^{(k)}) =: \text{grad}_\theta \mathcal{L}(\theta^{(k)})$ be the gradient of the training loss function at iteration k, then an efficient method for preconditioning is via a diagonal matrix B with entries

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

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

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

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

[1] http://jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

AdaGrad improvements: RMSProp and AdaDelta

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

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

AdaDelta (Zeiler 12'[2]) extends AdaGrad using a similar preconditioned as B_k^{RMS}, but also estimates the stepsize using an average difference in $\theta^{(k)} - \theta^{(k-1)}$.

[2] https://arxiv.org/abs/1212.5701

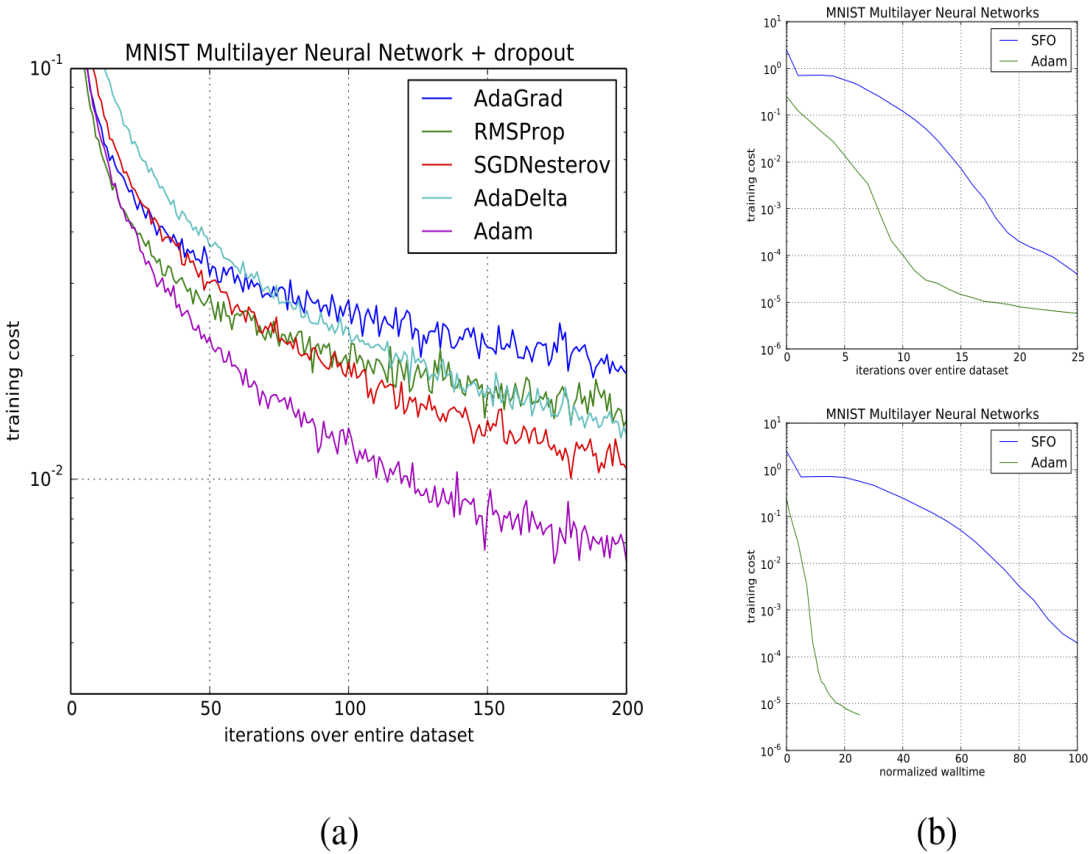
Adaptive moment estimation (Adam) (Kingma et al. 15'[3])

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 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd 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)

[3] https://arxiv.org/pdf/1412.6980.pdf

Adaptive moment estimation (Adam) (Kingma et al. 15'[4])



(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)

[4] https://arxiv.org/pdf/1412.6980.pdf

Adaptive moment estimation (Adam) (Kingma et al. 15'[5])

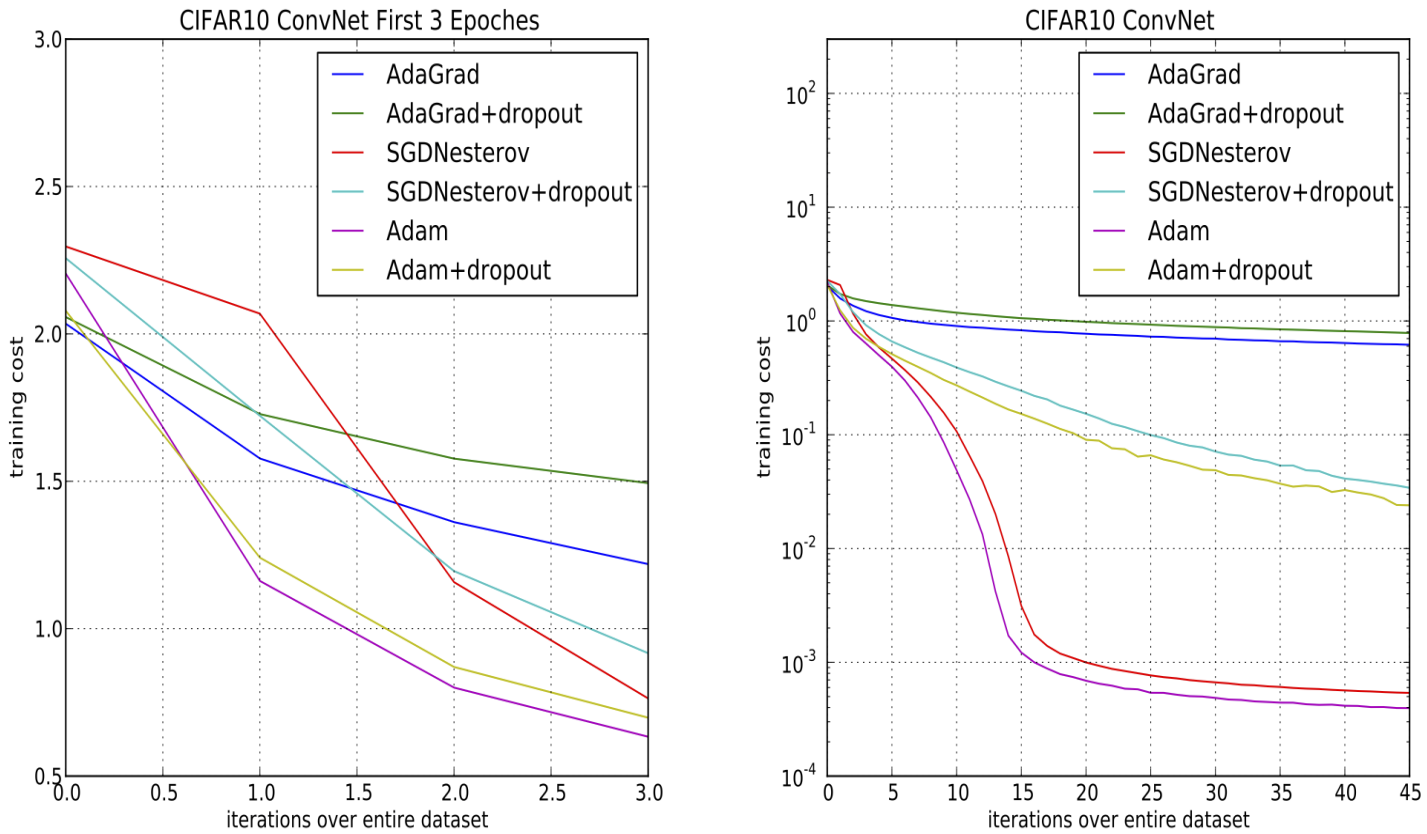


Figure 3: Convolutional neural networks training cost. (left) Training cost for the first three epochs. (right) Training cost over 45 epochs. CIFAR-10 with c64-c64-c128-1000 architecture.

[5] https://arxiv.org/pdf/1412.6980.pdf

AdaGrad: as adaptive stepsize rule (Ward et al. 18'[6])

Let $g^{(k)}(\theta^{(k)}) =: \text{grad}_\theta \mathcal{L}(\theta^{(k)})$ be the gradient of the training loss function at iteration k, then an efficient method for preconditioning is via a diagonal matrix B with entries

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

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

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

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

[6] https://arxiv.org/pdf/1806.01811.pdf

AdaGrad: scalar update in batch setting (Ward et al. 18'[7])

Given a loss function $\mathcal{L}(\theta; X, Y)$ consider the scalar AdaGrad update algorithm: Initialize with $\theta^{(0)}$ and $b_0 > 0$

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

Consider $\mathcal{L}(\theta) \in C_L^1$, that is L is the smallest value 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 held as a fixed constant b, then if $b < L/2$ gradient descent can diverge, while if $b \geq L$ then $T = 2b\epsilon^{-1}\mathcal{L}(\theta^{(0)})$.

[7] https://arxiv.org/pdf/1806.01811.pdf

AdaGrad: scalar proof ingredients (Ward et al. 18'[8])

The convergence complexity for AdaGrad in the scalar batch setting follows from the 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).

[8] https://arxiv.org/pdf/1806.01811.pdf

AdaGrad: scalar stochastic (Ward et al. 18'[9])

Let $g^{(k)}$ be an unbiased estimator of the gradient $\mathrm{grad}_\theta \mathcal{L}(\theta^{(k)})$ of the training loss function at iteration k; that is $\mathbb{E}(g^{(k)}) = \mathrm{grad}_\theta \mathcal{L}(\theta^{(k)})$. 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

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

[9] https://arxiv.org/pdf/1806.01811.pdf

AdaGrad: batch MNIST (Ward et al. 18'[10])

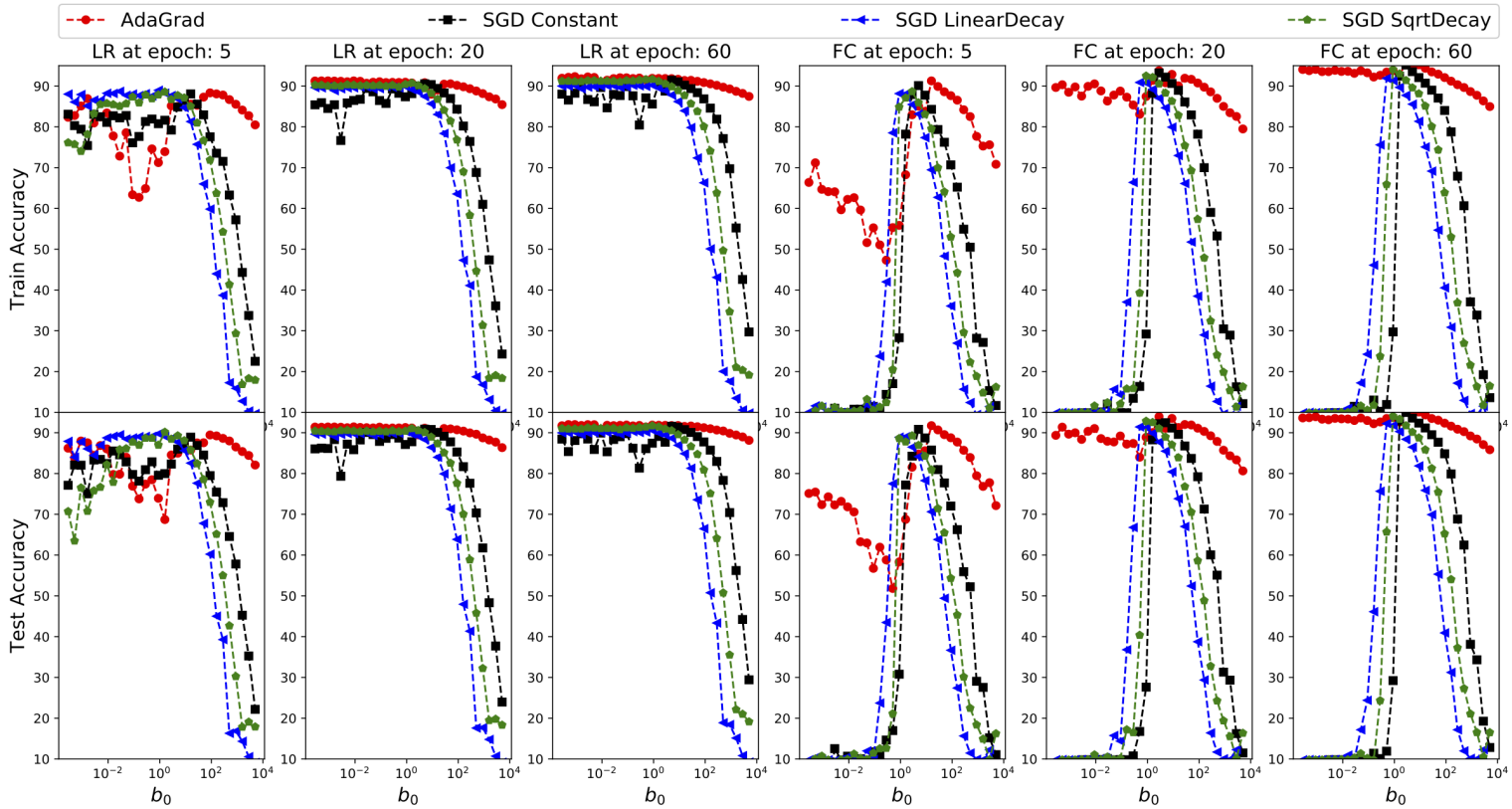


Figure 1: Batch setting on MNIST. Top (bottom) row are plots of train (test) accuracy with respect to the initialization b_0. The left 6 figures are for logistic regression (LR) with snapshots at epoch 5, 20 and 60 in the 1st, 2nd and 3rd column respectively. The right 6 figures are for two fully connected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

[10] https://arxiv.org/pdf/1806.01811.pdf

AdaGrad: stochastic MNIST (Ward et al. 18'[11])

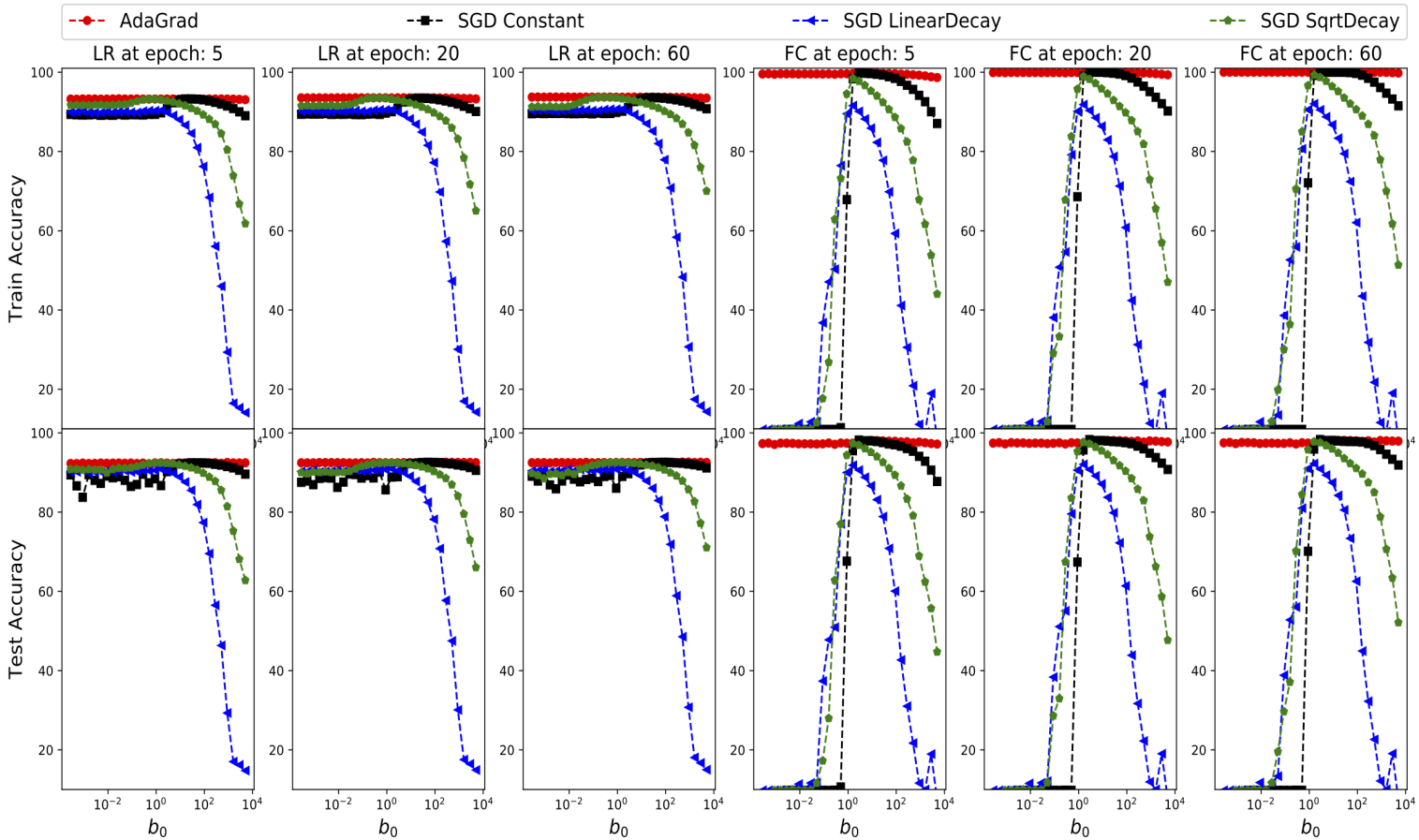


Figure 2: Stochastic setting on MNIST. Left 6 figures by logistic regression and right 6 figures by two fully connected layer. Note that the scale of y-axis change. See Figure 1 for reading instruction.

[11] https://arxiv.org/pdf/1806.01811.pdf

AdaGrad: stochastic CIFAR10 (Ward et al. 18'[12])

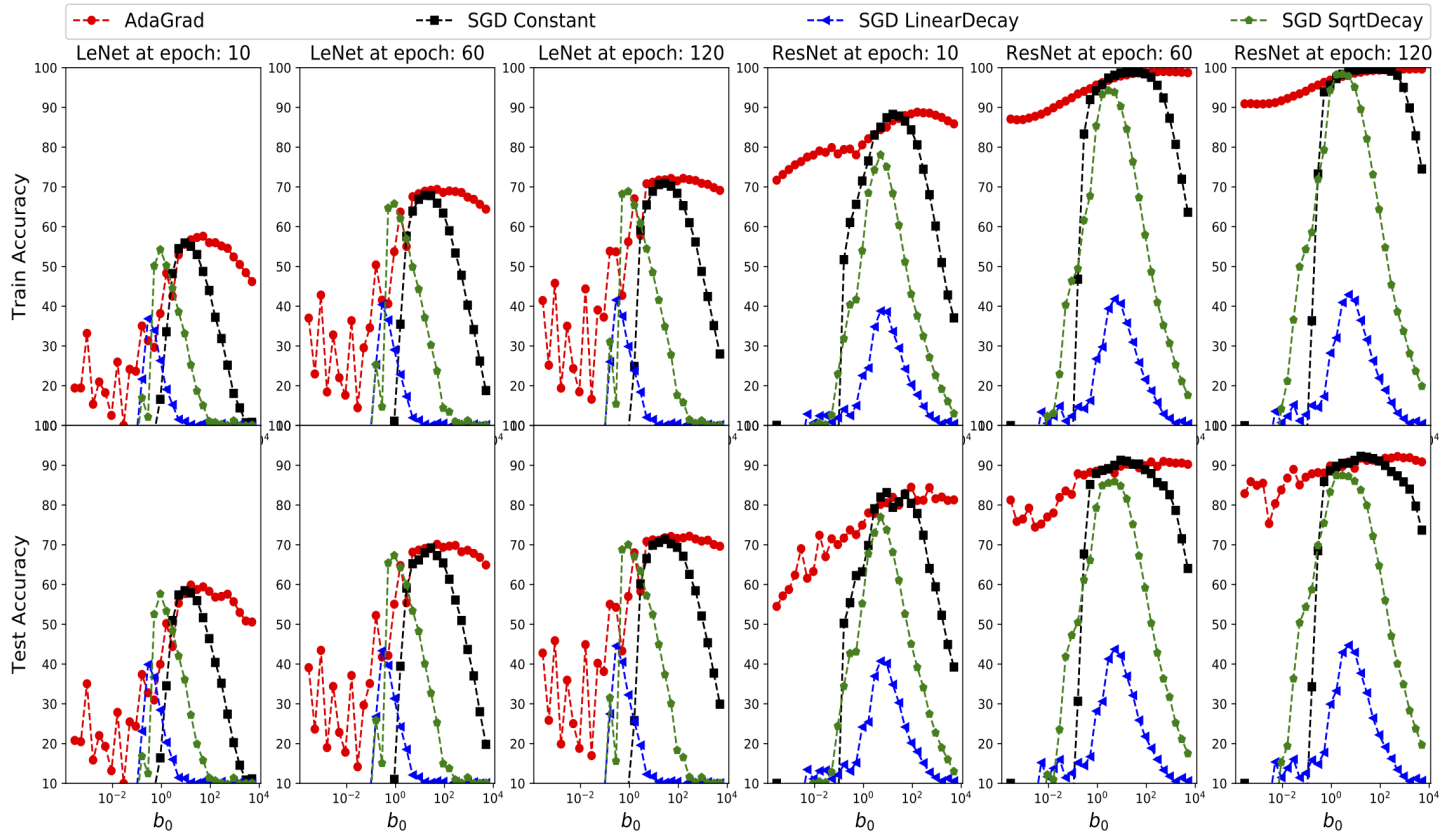


Figure 3: Stochastic setting on CIFAR10. Left 6 figures by LeNet and right 6 figures by ResNet. Note that the epoch (see title) is different from previous figures and no momentum is used. See Figure 1 for reading instruction.

[12] https://arxiv.org/pdf/1806.01811.pdf