Enhanced Momentum based Variants of Gradient Descent

George Michailidis

gmichail@ucla.edu

STAT 102B

Key take home message from Lecture 2.1

Polyak and especially Nesterov momentum methods mitigate difficulties in making "good progress" that gradient descent faces with optimization problems where the curvature of the function greatly differs across coordinates

With new machine and especially deep learning models that lead to more challenging optimization problems, new momentum based methods have been developed in the last 12 years

Note that the enhanced momentum methods presented next, do not use a descent direction in every iteration (similarly to Polyak and Nesterov)

Adaptive Gradient Descent (AdaGrad) - I

The standard gradient descent update uses a step size η_k for all the coordinates of $\nabla f(x)$

Question:
Can we make the step size adaptive to each coordinate of the gradient?

Intuitively, we would like to design a vector of step sizes, one for each coordinate of the gradient; i.e.,

$$\vec{\tilde{\eta}}_k = (\tilde{\eta}_k^1, \tilde{\eta}_k^2, \cdots, \tilde{\eta}_k^n)^\top$$

Adaptive Gradient Descent (AdaGrad) - II

Denote by $G_k \equiv \nabla f(x_k)$; i.e., the gradient evaluated at the current update x_k

Let $\odot$ denote the Hadamard product for two vectors $u, v \in \mathbb{R}^n$ and defined as

$$w = u \odot v \Longrightarrow w(i) = u(i)v(i), \ i = 1, \cdots, n \ \text{(element-wise multiplication)} \quad (1)$$

For example, let

$$u = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \ v = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

Then,

$$w = u \odot v = \begin{bmatrix} 1 \times 3 \\ 2 \times 4 \end{bmatrix} = \begin{bmatrix} 3 \\ 8 \end{bmatrix}$$

Adaptive Gradient Descent (AdaGrad) - III

Let

$$z_k = z_{k-1} + G_k \odot G_k \tag{2}$$

with $z_0 = \epsilon > 0$ very small; e.g., all elements of the vector z_0 set to 10^{-6} or smaller

Denote by $\tilde{z}_k$ the *n*-dimensional vector whose *i*-th element is the reciprocal of the square root of $z_k(i)$; namely

$$\tilde{z}_k(i) = 1/\sqrt{z_k(i)}, \quad i = 1, \cdots, n \tag{3}$$

Then, the AdaGrad updated is given by

$$x_{k+1} = x_k - \eta\big(\tilde{z}_k \odot G_k\big) \tag{4}$$

where $\eta > 0$ is set a priori to a small value; e.g., $\eta = 0.05$ or 0.01

Remarks on AdaGrad

Note that

$$z_k = z_{k-1} + G_k \odot G_k = \cdots = \sum_{i=0}^{k} G_i \odot G_i \tag{5}$$

and hence it utilizes the whole history of gradient updates

One issue with AdaGrad is that over iterations the "effective" step size vector $\tilde{\eta}_k = \eta \tilde{z}_k$ may become too small and hence may slow down convergence

Adaptive Movement Estimation (ADAM) - I

Combines momentum and adaptive step sizes to improve convergence of gradient descent

Let

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) G_k, \quad \beta_1 \in (0, \beta_2) \tag{6}$$

with $m_0 = 0$ and $G_k \equiv \nabla f(x_k)$

(6) is referred to as the first moment equation

Note that (6) computes a moving average of past gradients

Modify (2) as follows:

$$z_k = \beta_2 z_{k-1} + (1-\beta_2)\big(G_k \odot G_k\big), \quad \beta_2 \in (0,1) \tag{7}$$

with $z_0 = \epsilon$ very small

(7) is referred to as the second moment equation

Adaptive Movement Estimation (ADAM) - II

Compute

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k}, \quad \hat{z}_k = \frac{z_k}{1-\beta_2^k} \tag{8}$$

(8) is referred to as the bias correction step

Compute $\tilde{z}_k(i) \equiv 1/\sqrt{\hat{z}_k(i)},\ i = 1, \cdots, n$ as before

Then, the ADAM update is given by

$$x_{k+1} = x_k - \eta\left(\tilde{z}_k \odot \hat{m}_k\right) \tag{9}$$

Recap - ADAM algorithm

Update Rules: ADAM

$$
\begin{aligned}
m_k &= \beta_1 m_{k-1} + (1-\beta_1) G_k && \text{(1st moment)} \\
z_k &= \beta_2 z_{t-1} + (1-\beta_2)\big(G_k \odot G_k\big) && \text{(2nd moment)} \\
\hat{m}_k &= \frac{m_k}{1-\beta_1^t}, \qquad \hat{z}_k = \frac{z_k}{1-\beta_2^t} && \text{(bias correction)} \\
\tilde{z}_k(i) &= 1/\sqrt{\hat{z}_k(i)} + \epsilon \\
x_{k+1} &= x_k - \eta\big(\tilde{z}_k \odot \hat{m}_k\big), \ \eta > 0
\end{aligned}
$$

Typical values for the hyper-parameters: $\beta_1 = 0.9, \beta_2 = 0.999$

Remarks on ADAM

1. ADAM is the default algorithm in PyTorch and Tensorflow for optimizing the parameters of deep neural networks (from multi-layer perceptrons, to recurrent neural networks, to convolutional neural networks, to transformers)
2. The original version from the 2014 paper can fail to converge even for "nice" functions (e.g., convex), and hence a number of variants have become available in PyTorch and Tensorflow
3. Coupling ADAM with Nesterov momentum (N-ADAM; `torch.optim.NAdam` is also an option in PyTorch and Tensorflow

AMSGrad Modification of ADAM - I

ADAM can fail to converge in some convex settings due to the adaptive learning rate increasing

The AMSGrad modification enforces a non-increasing step size by modifying the second-moment equation

Update Rules: AMSGrad-ADAM

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) G_k \qquad \text{(1st moment)}$$

$$z_k = \beta_2 z_{t-1} + (1-\beta_2)\big(G_k \odot G_k\big) \qquad \text{(2nd moment)}$$

$$\hat{m}_k = \frac{m_k}{1-\beta_1^t}, \qquad \text{(bias correction only for 1st moment)}$$

$$\hat{z}_k = \max\{\hat{z}_{k-1}, z_k\} \qquad \text{(replaces bias correction for 2nd moment)}$$

$$\tilde{z}_k(i) = 1/\sqrt{\hat{z}_k(i)} + \epsilon$$

$$x_{k+1} = x_k - \eta\big(\tilde{z}_k \odot \hat{m}_k\big), \ \eta > 0$$

AMSGrad Modification of ADAM - II

It can be seen that this modification uses the maximum of all historical z_k terms to prevent adaptive step sizes from increasing

In PyTorch, one can specify this version of ADAM by
`torch.optim.Adam(amsgrad=True)`

ADAM-W Modification - I

Key idea: introduce a weight decay in the update[1]

Update Rule: ADAM-W

$$x_{k+1} = (1 - \eta\lambda)x_k - \eta\left(\tilde{z}_k \odot \hat{m}_k\right),$$

In PyTorch, one can specify this version of ADAM by
`torch.optimW(weight_decay=1e-02)`

[1] a more in depth explanation of the key ideas behind ADAM and ADAM-W will be given after Stochastic Gradient Descent is presented

ADAM-W Modification - II

Task / Model Type	Typical $\boldsymbol{\lambda}$ (weight decay)
Vision (CNNs, ViTs)	10^{-4} to 10^{-2}
Transformers	0.01
NLP (non-transformer)	10^{-5} to 10^{-3}
Small models / overfitting-prone	Up to 10^{-2}
Large models / well-regularized	10^{-5} or lower

Table 1: Recommended weight decay λ values for ADAM-W across different tasks and model types

Revisiting Motivating Example III from Lecture 2.1

Consider the following quadratic function $f : \mathbb{R}^2 \to \mathbb{R}$,

$$f(x) = \frac{1}{2} x^\top Q x$$

with

$$Q = \begin{bmatrix} 2.505 & -2.495 \\ -2.495 & 2.505 \end{bmatrix}$$

$\lambda_{\max}(Q) = 5$

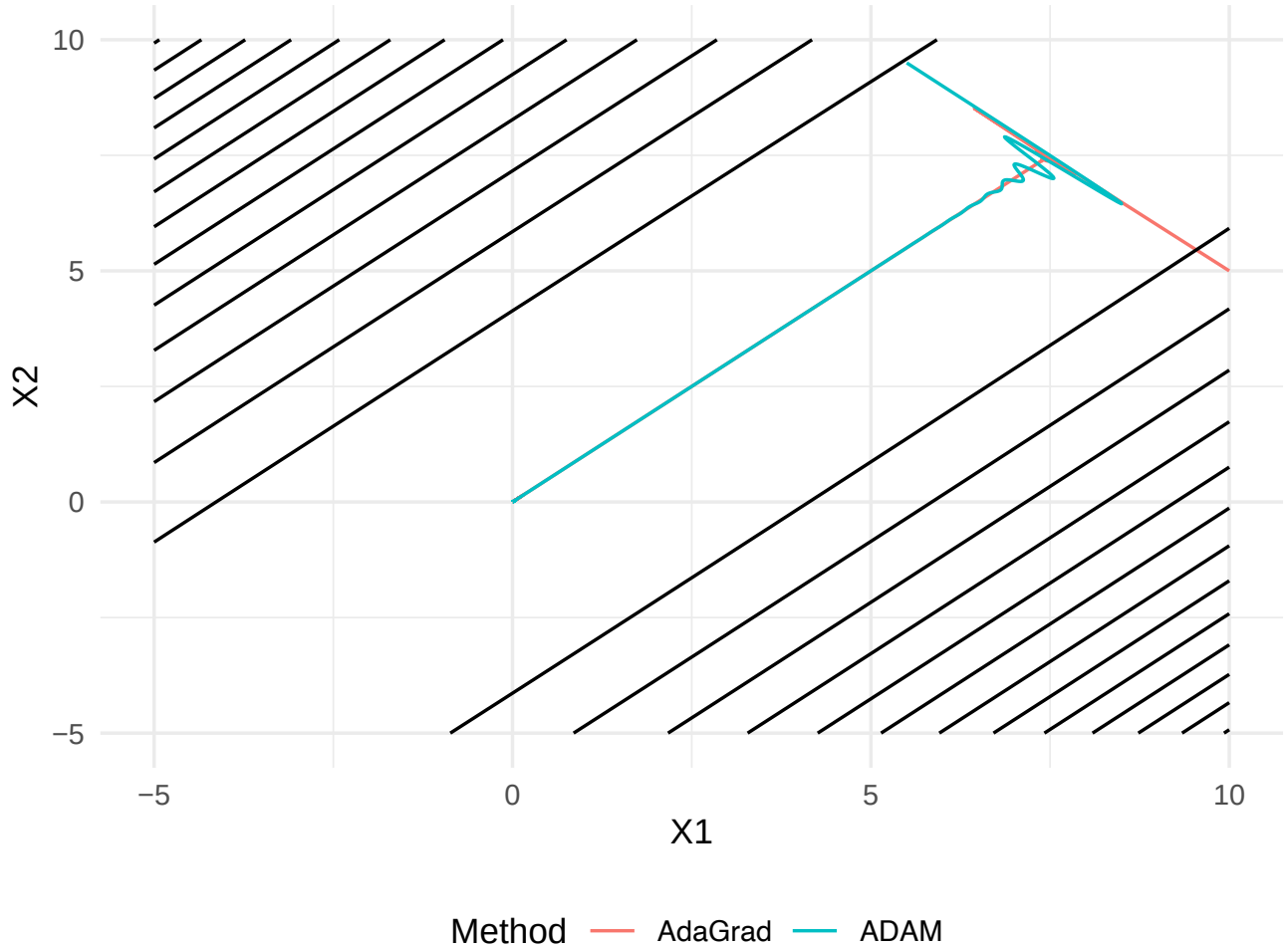

Figure 1: Adagrad and ADAM iterates, with $\eta = 5, 0.5$ (Adagrad, ADAM), # iterations $\approx 1200, 800$ (Adagrad, ADAM)

Remarks

It can be seen that for this simple, but ill-conditioned quadratic function,

- the enhanced momentum methods offer modest improvements in terms of number of iterations needed for obtaining an accurate solution, compared to standard gradient descent;
- however, they require many more iterations that simpler momentum methods (Polyak, Nesterov)

These enhanced momentum methods are most advantageous for more complex optimization problems and for the stochastic version of gradient descent (Stochastic Gradient Descent)

Motivating Example III with ADAM-W

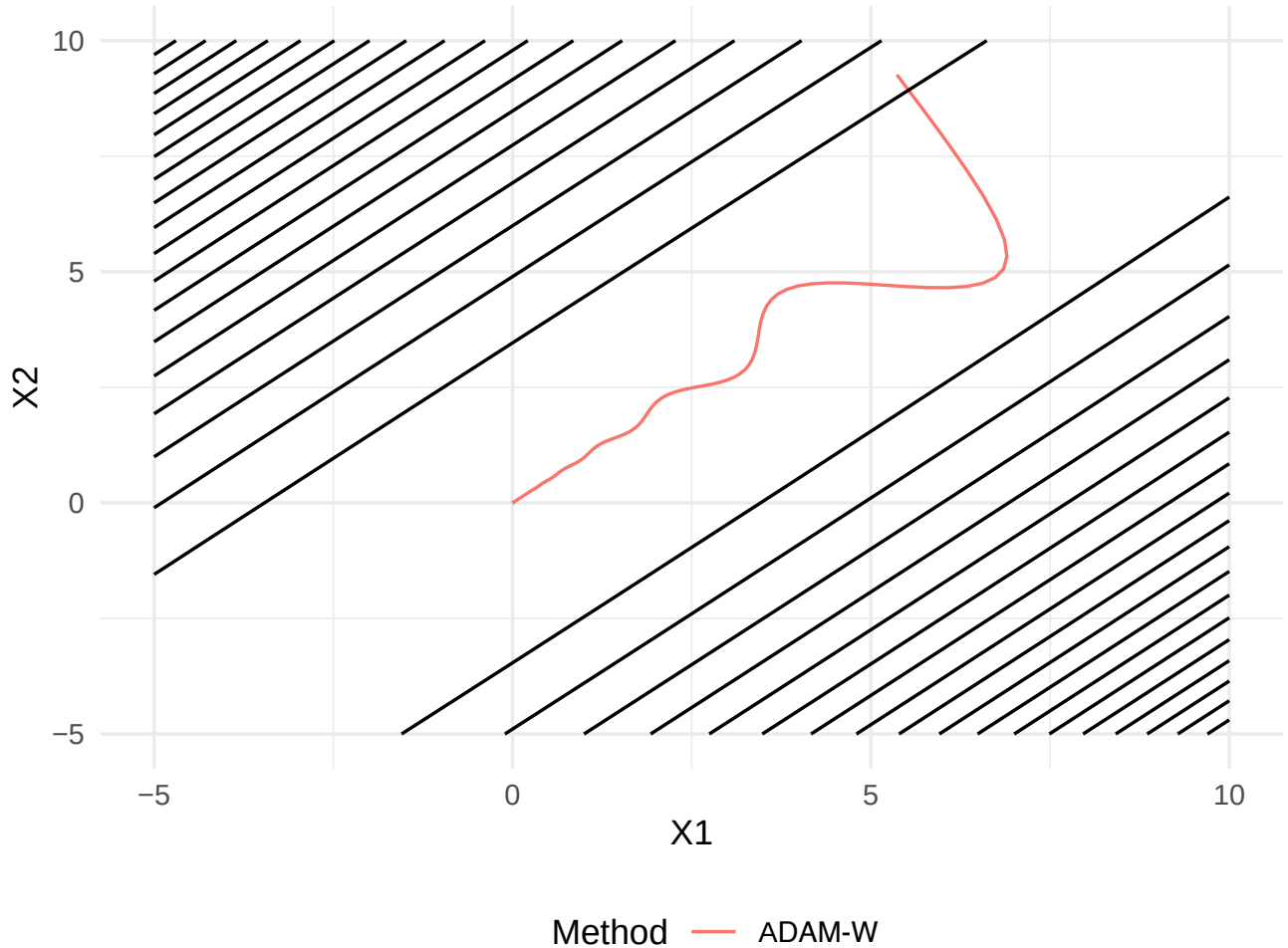

Figure 2: ADAM-W iterates, with $\lambda = 0.5$ and $\eta = 5, 0.5$, # iterations 283