11.5. Accelerated Gradient Methods

Lecture based on "Dive into Deep Learning" **http://D2L.AI** (Zhang et al., 2020)

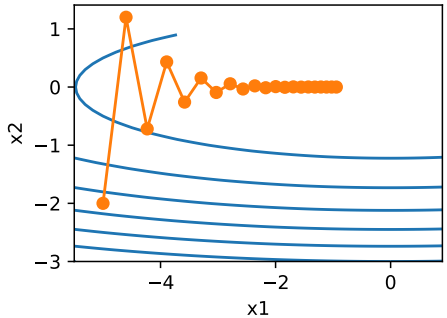
Prof. Dr. Christoph Lippert

Digital Health & Machine Learning

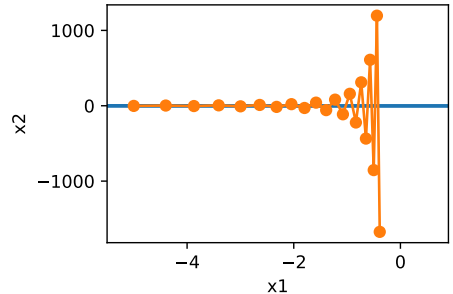
Getting the learning rate right is crucial

Example

$$f(\boldsymbol{x}) = 0.1x_1^2 + 2x_2^2$$



learning rate 0.4.



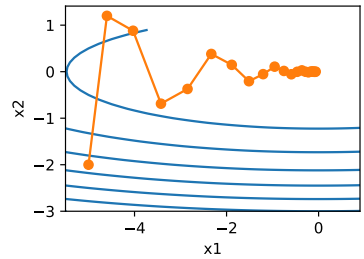
learning rate 0.6.

The **momentum** method solves this problem introducing a velocity $\mathbf{v}_t$ at each step t.

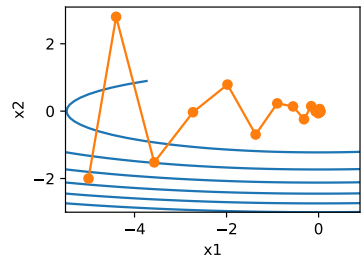
$$\mathbf{v}_t \leftarrow \gamma \mathbf{v}_{t-1} + \eta_t \mathbf{g}_t,$$
$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{v}_t,$$

- At $t = 0$, momentum initializes the velocity variable $\mathbf{v}_0 = \vec{0}$
- the **momentum hyperparameter** γ satisfies $0 \leq \gamma < 1$.
  When $\gamma = 0$, momentum is equivalent to a mini-batch SGD.

$\eta = 0.4, \quad \gamma = 0.5$



$\eta = 0.6, \quad \gamma = 0.5$

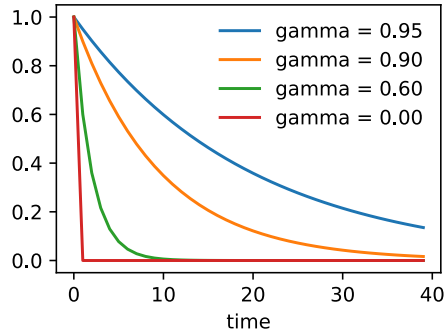


To understand momentum, we expand the velocity variable over time:

$$\begin{aligned}\mathbf{v}_t =& \eta_t \mathbf{g}_t + \gamma \mathbf{v}_{t-1},\\ =& \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \gamma \mathbf{v}_{t-1},\\ & \dots\\ =& \eta_t \mathbf{g}_t + \gamma \eta_{t-1} \mathbf{g}_{t-1} + \ldots + \gamma^{t-1} \eta_1 \mathbf{g}_1.\end{aligned}$$

- $\mathbf{v}_t$ is a weighted sum over all past gradients multiplied by the according learning rate
- which is the weight update in normal gradient descent.
- The weights deceases exponentially with the speed controlled by γ.

weights for 40 time steps under various γs.



- Momentum smooths the weight updates over time.
- A small γ will focus on more recent scaled gradients.
- A large γ will include more past scaled gradients.
- It might walk out of a region that is flat.

Note that $\frac{1}{1-\gamma} = 1 + \gamma + \gamma^2 + \cdots$.

- If all scaled gradients are similar to each other e.g. $\eta_t \mathbf{g}_t \approx \eta \mathbf{g}$ for all ts,
- then the momentum changes the weight updates from $\eta \mathbf{g}$ in gradient descent to $\frac{\eta}{1-\gamma} \mathbf{g}$.

Summary

- The momentum method uses an exponentially weighted moving average (EWMA) of the gradient.
  It takes the weighted average of past time steps, with weights that decay exponentially by the time step.
- Momentum smooths independent variable updates for adjacent time steps in direction.

- In DL we apply a learning rate decay. (e.g. $\eta_t = \frac{\eta_0}{\sqrt{t+c}}$)
- If we observe gradient information in some dimensions only infrequently
  - e.g., in a language model where we oberve some words only rarely (sparse features)
  - More generally, we have small partial derivatives in some dimensions most of the time and only rarely large ones
  - then the learning rate for this dimension might have already decaeyed too much for this dimension to still make progress
- Adagrad addresses this issue by tracking the aggregate of $(\partial_i f(\mathbf{x}))^2$ for each dimension.

Example

$f(\vec{x})$ with $\vec{x} = [x_1, x_2]^\top$, each element x_1 and x_2 uses the same learning rate η:

$$x_1 \leftarrow x_1 - \eta \frac{\partial f}{\partial x_1}, \quad x_2 \leftarrow x_2 - \eta \frac{\partial f}{\partial x_2}.$$

- When there is a big difference in gradient values for x_1 and x_2, a small learning rate is needed to avoid divergence in larger gradient dimensions.
- This will cause the independent variables to move slowly in dimensions with smaller gradient values.

Adagrad adjusts the learning rate according to the gradient value in each dimension.

Adagrad re-adjusts the learning rate of each parameter using the cumulative variable $\boldsymbol{s}_t$ capturing the square of the gradient dimensions:

$$\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t,$$

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t,$$

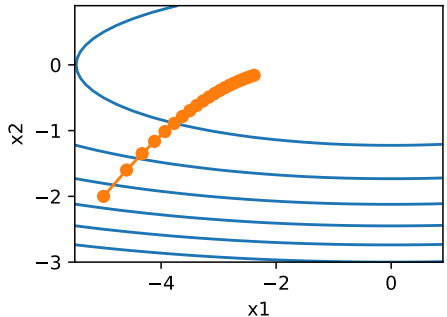
- $\odot$ is element-wise multiplication.
- The square root and division, and multiplication operations are all element operations.
- η is the learning rate
- ϵ is a constant to maintain numerical stability, such as 10^{-6}.
- At time step 0, Adagrad initializes each element in $\boldsymbol{s}_0$ to 0.

$$\boldsymbol{s}_t \leftarrow \boldsymbol{s}_{t-1} + \boldsymbol{g}_t \odot \boldsymbol{g}_t,$$
$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \frac{\eta}{\sqrt{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t,$$

- For elements that have a constant and large partial derivative, the learning rate will drop faster.
- If the partial derivative remains small, then its learning rate will decline more slowly.
- The learning rate of each element declines during each iteration.
- The learning rate declines very fast during an early iteration.
- Adagrad might have difficulty finding an optimum because the learning rate will be too small at later iterations.

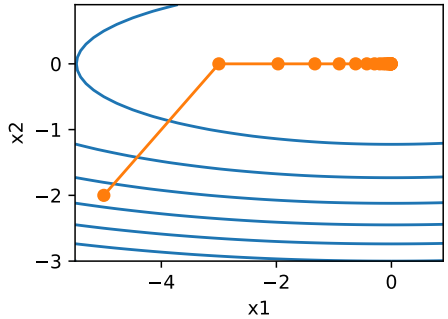
$$f(\boldsymbol{x}) = 0.1x_1^2 + 2x_2^2$$

learning rate 0.4



- the trajectory is smoother
- Due to the cumulative effect of $\boldsymbol{s}_t$, the learning rate continuously decays

learning rate 2



- the independent variable approaches the optimum more quickly.

Summary

- Adagrad constantly adjusts the learning rate during each iteration to give each element in the independent variable of the objective function its own learning rate.
- When using Adagrad, the learning rate of each element in the independent variable decreases during each iteration.
- Adagrad might have difficulty finding a useful solution because the learning rate will be too small at later iterations.

In **RMSProp** the entries of $\mathbf{s}_t$ are computed as EWMA of the squares of the gradient elements.

$$\mathbf{s}_t \leftarrow \gamma \mathbf{s}_{t-1} + (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t.$$

RMSProp re-adjusts the learning rate as in Adagrad.

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \odot \mathbf{g}_t,$$

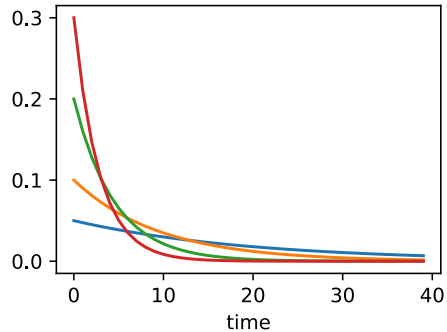
- $0 \leq \gamma < 1$ is a hyperparameter of the EWMA
- η is the learning rate
- ϵ is a constant added to maintain numerical stability, such as 10^{-6}.

Expanding the definition of $\mathbf{s}_t$

$$\begin{aligned}\mathbf{s}_t &= (1-\gamma)\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{s}_{t-1} \\ &= (1-\gamma)\left(\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{g}_{t-1} \odot \mathbf{g}_{t-1}\right) + \gamma^2 \mathbf{s}_{t-2} \\ &\dots \\ &= (1-\gamma)\left(\mathbf{g}_t \odot \mathbf{g}_t + \gamma \mathbf{g}_{t-1} \odot \mathbf{g}_{t-1} + \dots + \gamma^{t-1}\mathbf{g}_1 \odot \mathbf{g}_1\right).\end{aligned}$$

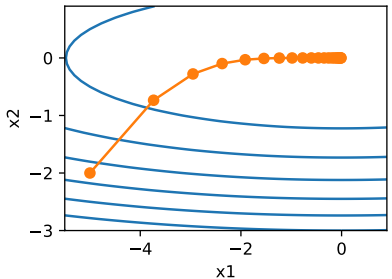
- $\frac{1}{1-\gamma} = 1 + \gamma + \gamma^2 + \dots$, so the sum of weights equals to 1.
- The weights decrease exponentially

Weights in the past 40 time steps with $\gamma \in [0.95, 0.9, 0.8, 0.7]$.

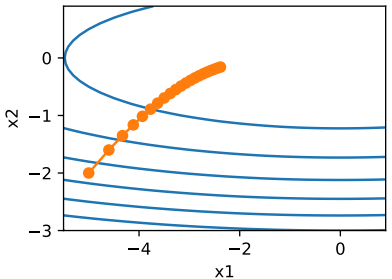


$$f(\mathbf{x}) = 0.1x_1^2 + 2x_2^2$$

$\eta = 0.4, \gamma = 0.9$



Adagrad with $\eta = 0.4$



- In contrast to Adagrad RMSProp can approach the optimal solution faster at $\eta = 0.4$.
- The difference between **RMSProp** and **Adagrad** is that RMSProp uses an EWMA on the squares of elements to adjust the learning rate.

Adadelta keeps track of two EWMAs, $\boldsymbol{s}_t$ and $\Delta\boldsymbol{x}_t$.

$$\boldsymbol{s}_t \leftarrow \rho \boldsymbol{s}_{t-1} + (1-\rho)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$$

$\Delta\boldsymbol{x}_{t-1}$ is used to compute the step $\boldsymbol{g}'_t$:

$$\boldsymbol{g}'_t \leftarrow \sqrt{\frac{\Delta\boldsymbol{x}_{t-1} + \epsilon}{\boldsymbol{s}_t + \epsilon}} \odot \boldsymbol{g}_t,$$

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{g}'_t.$$

$\Delta\boldsymbol{x}$ records the EWMA on the squares of the steps $\boldsymbol{g}'$

$$\Delta\boldsymbol{x}_t \leftarrow \rho\Delta\boldsymbol{x}_{t-1} + (1-\rho)\boldsymbol{g}'_t \odot \boldsymbol{g}'_t.$$

- $0 \leq \rho < 1$ (counterpart of γ in RMSProp)
- ϵ is a constant added to maintain the numerical stability, such as 10^{-5}.
- Adadelta replaces the learning rate η in RMSProp with $\sqrt{\Delta\boldsymbol{x}_{t-1}}$.

Adam uses the **momentum** $\boldsymbol{v}_t$ as the EWMA over the gradient.

$$\boldsymbol{v}_t \leftarrow \beta_1 \boldsymbol{v}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t.$$

$\boldsymbol{s}_t$ is the EWMA on the squares of elements in the gradient

$$\boldsymbol{s}_t \leftarrow \beta_2 \boldsymbol{s}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t \odot \boldsymbol{g}_t.$$

- $0 \leq \beta_1 < 1$ (the author suggests 0.9),
- $0 \leq \beta_2 < 1$ (the author suggests 0.999)

Because of initialization, $\boldsymbol{v}_0 = 0$ and $\boldsymbol{s}_0 = 0$,

$$\boldsymbol{v}_t = (1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} \boldsymbol{g}_i$$

For example, when $\beta_1 = 0.9$, $\boldsymbol{v}_1 = 0.1\boldsymbol{g}_1$.

- The sum of weights from all previous time steps is $(1 - \beta_1) \sum_{i=1}^{t} \beta_1^{t-i} = 1 - \beta_1^t$.
- For small t, the sum will be small.

Adam corrects for this bias in $\boldsymbol{v}_t$ and $\boldsymbol{s}_t$:

$$\hat{\boldsymbol{v}}_t \leftarrow \frac{\boldsymbol{v}_t}{1 - \beta_1^t}, \qquad \hat{\boldsymbol{s}}_t \leftarrow \frac{\boldsymbol{s}_t}{1 - \beta_2^t}.$$

Adam uses the bias-corrected variables $\hat{\boldsymbol{v}}_t$ and $\hat{\boldsymbol{s}}_t$ to re-adjust the learning rate of each model parameter

$$\boldsymbol{g}_t' \leftarrow \frac{\eta \hat{\boldsymbol{v}}_t}{\sqrt{\hat{\boldsymbol{s}}_t} + \epsilon},$$

- η is the learning rate
- ϵ is a constant added to maintain numerical stability, such as 10^{-8}.

Finally, use $\boldsymbol{g}_t'$ to update the parameters:

$$\boldsymbol{x}_t \leftarrow \boldsymbol{x}_{t-1} - \boldsymbol{g}_t'.$$

Summary

- Created on the basis of RMSProp, Adam also uses EWMA on the mini-batch stochastic gradient
- Adam uses bias correction.

Summary

- Most obviously the **magnitude** of the learning rate matters.
  - If it is too large, optimization diverges
  - If it is too small, it takes too long to train or we end up with a suboptimal result.
- The rate of decay is just as important.
  - If the learning rate remains large we may simply end up bouncing around the minimum and thus not reach optimality.
  - In short, we want the rate to decay, but probably more slowly than $\mathcal{O}(t^{-\frac{1}{2}})$ which would be a good choice for convex problems.
- Another aspect that is equally important is **initialization**. This pertains both to how the parameters are set initially and also how they evolve initially.