Machine Learning for Computer Vision

Last update: 23 September 2024

Academic Year 2024 – 2025
Alma Mater Studiorum · University of Bologna

Contents

1 Optimizers

1.1 Stochastic gradient descent with mini-batches

SGD

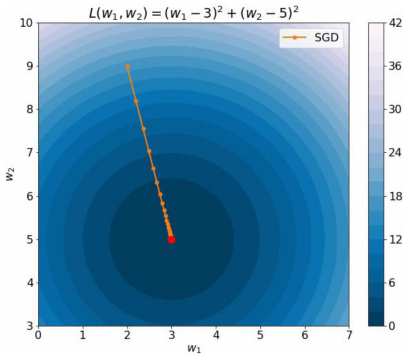
Stochastic gradient descent (SGD) Gradient descent based on a noisy approximation of the gradient computed on mini-batches of B data samples.

An epoch e of SGD with mini-batches of size B does the following:

1. Shuffle the training data $\boldsymbol{D}^{\text{train}}$.
2. For $u = 0, \dots, U$, with $U = \lceil \frac{N}{B} \rceil$:
   a) Classify the examples $\boldsymbol{X}^{(u)} = \{\mathbf{x}^{(Bu)}, \dots, \mathbf{x}^{(B(u+1)-1)}\}$ to obtain the predictions $\hat{Y}^{(u)} = f(\mathbf{X}^{(u)}; \boldsymbol{\theta}^{(e*U+u)})$ and the loss $\mathcal{L}\big(\boldsymbol{\theta}^{(e*U+u)}, (\boldsymbol{X}^{(u)}, \hat{Y}^{(u)})\big)$.
   b) Compute the gradient $\nabla\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\big(\boldsymbol{\theta}^{(e*U+u)}, (\boldsymbol{X}^{(u)}, \hat{Y}^{(u)})\big)$.
   c) Update the parameters $\boldsymbol{\theta}^{(e*U+u+1)} = \boldsymbol{\theta}^{(e*U+u)} - \texttt{lr} \cdot \nabla\mathcal{L}$.

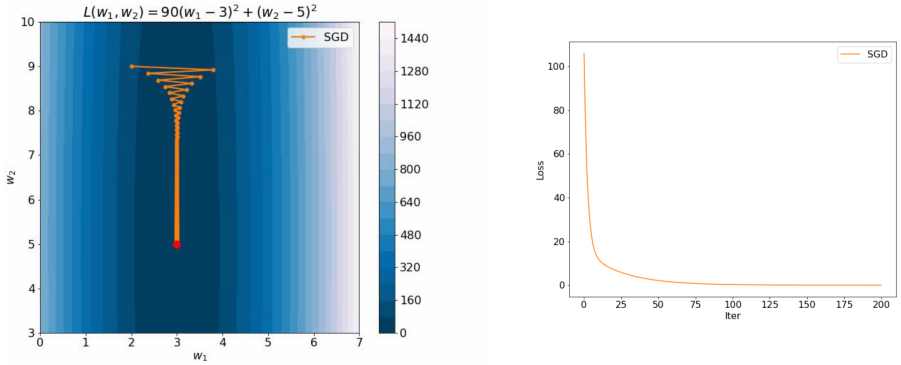
SGD spheres

Remark (Spheres)**.** GD/SGD works better on convex functions (e.g., paraboloids) as there are no preferred directions to reach a minimum. Moreover, faster convergence can be obtained by using a higher learning rate.

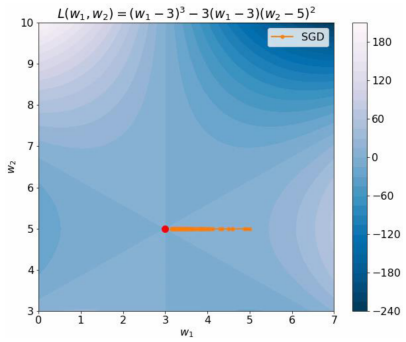


SGD canyons

Remark (Canyons)**.** A function has a canyon shape if it grows faster in some directions. The trajectory of SGD oscillates in a canyon (the steep area) and a smaller learning rate is required to reach convergence. Note that, even though there are oscillations, the loss alone decreases and is unable to show the oscillating behavior.



Remark (Local minima)**.** GD/SGD converges to a critical point. Therefore, it might end up in a saddle point or local minima. SGD local minima



1.2 Second-order methods

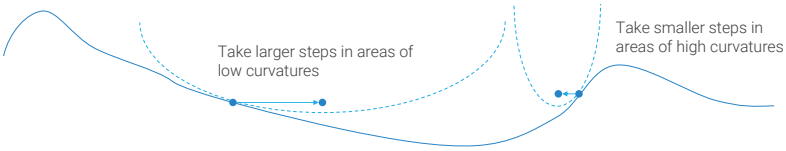
Methods that also consider the second-order derivatives when determining the step.

Newton's method Second-order method for the 1D case based on the Taylor expansion: Newton's method

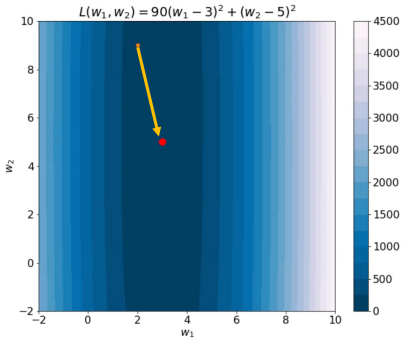
$$f(x_t + \Delta x) \approx f(x_t) + f'(x_t)\Delta x + \frac{1}{2} f''(x_t)\Delta x^2$$

which can be seen as a paraboloid over the variable Δx.

Given a function f and a point x_t, the method fits a paraboloid at x_t with the same slope and curvature at $f(x_t)$. The update is determined as the step required to reach the minimum of the paraboloid from x_t. It can be shown that this step is $-\frac{f'(x_t)}{f''(x_t)}$.



Remark. For quadratic functions, second-order methods converge in one step.



General second-order method For a generic multivariate non-quadratic function, the update is: General second-order method

$$-\texttt{lr} \cdot \boldsymbol{H}_f^{-1}(\mathbf{x}_t)\nabla f(\mathbf{x}_t)$$

where $\boldsymbol{H}_f$ is the Hessian matrix.

Remark. Given k variables, $\boldsymbol{H}$ requires $O(k^2)$ memory. Moreover, inverting a matrix has time complexity $O(k^3)$. Therefore, in practice second-order methods are not applicable for large models.

1.3 Momentum

Standard momentum Add a velocity term $v^{(t)}$ to account for past gradient updates: Standard momentum

$$v^{(t+1)} = \mu v^{(t)} - \texttt{lr}\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + v^{(t+1)}$$

where $\mu \in [0, 1[$ is the momentum coefficient.

In other words, $v^{(t+1)}$ represents a weighted average of the updates steps done up until time t.

Remark. Momentum helps to counteract a poor conditioning of the Hessian matrix when working with canyons.

Remark. Momentum helps to reduce the effect of variance of the approximated gradients (i.e., acts as a low-pass filter).

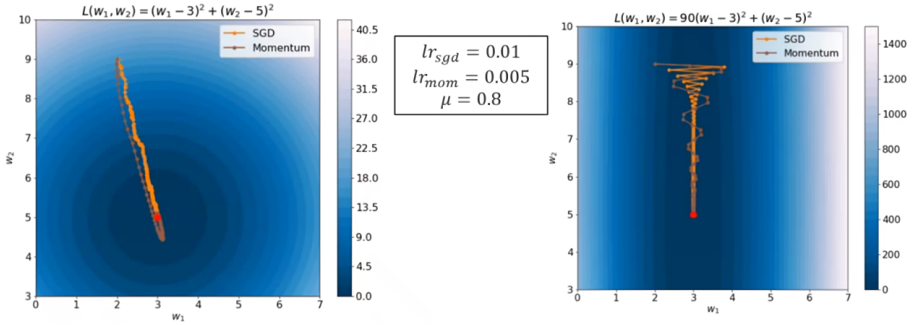


Figure 1.1: Plain SGD vs momentum SGD in a sphere and a canyon. In both cases, momentum converges before SGD.

Nesterov momentum Variation of the standard momentum that computes the gradient step considering the velocity term: Nesterov momentum

$$v^{(t+1)} = \mu v^{(t)} - \texttt{lr}\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)} + \mu v^{(t)})$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + v^{(t+1)}$$

Remark. The key idea is that, once $\mu v^{(t)}$ is summed to $\boldsymbol{\theta}^{(t)}$, the gradient computed at $\boldsymbol{\theta}^{(t)}$ is obsolete as $\boldsymbol{\theta}^{(t)}$ has been partially updated.

Remark. In practice, there are methods to formulate Nesterov momentum without the need of computing the gradient at $\boldsymbol{\theta}^{(t)} + \mu v^{(t)}$.

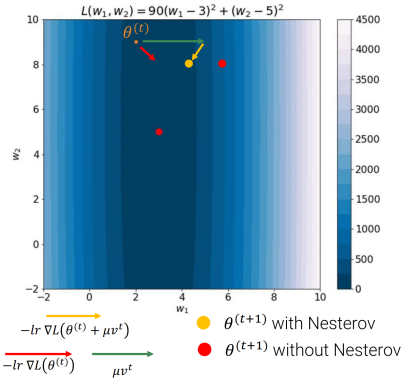


Figure 1.2: Visualization of the step in Nesterov momentum

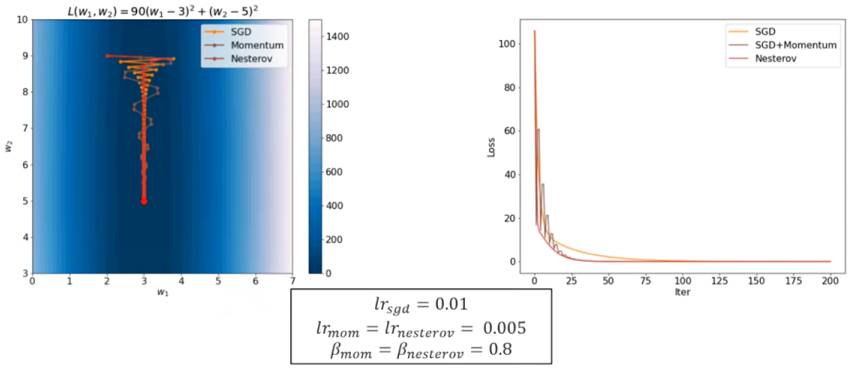


Figure 1.3: Plain SGD vs standard momentum vs Nesterov momentum

1.4 Adaptive learning rates methods

Adaptive learning rates Methods to define per-parameter adaptive learning rates.

Adaptive learning rates

Ideally, assuming that the changes in the curvature of the loss are axis-aligned (e.g., in a canyon), it is reasonable to obtain a faster convergence by:

- Reducing the learning rate along the dimension where the gradient is large.
- Increasing the learning rate along the dimension where the gradient is small.

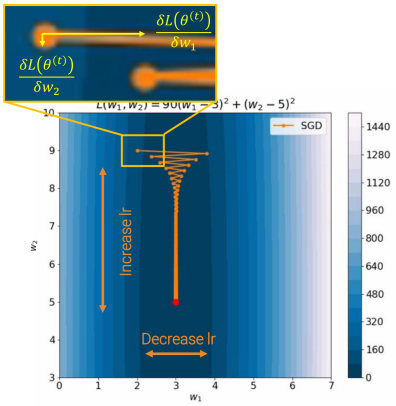


Figure 1.4: Loss where the w_1 parameter has a larger gradient, while w_2 has a smaller gradient

Remark. As the landscape of a high-dimensional loss cannot be seen, automatic methods to adjust the learning rates must be used.

1.4.1 AdaGrad

Adaptive gradient (AdaGrad) Each entry of the gradient is rescaled by the inverse of the history of its squared values: Adaptive gradient (AdaGrad)

$$\mathbb{R}^{N\times 1} \ni \mathbf{s}^{(t+1)} = \mathbf{s}^{(t)} + \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}) \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\mathbb{R}^{N\times 1} \ni \boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\texttt{lr}}{\sqrt{\mathbf{s}^{(t+1)}} + \varepsilon} \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$

where:

- $\odot$ is the element-wise product.
- Division and square root are element-wise.
- ε is a small constant.

Remark. By how it is defined, $\mathbf{s}^{(t)}$ is monotonically increasing which might reduce the learning rate too early when the minimum is still far away.

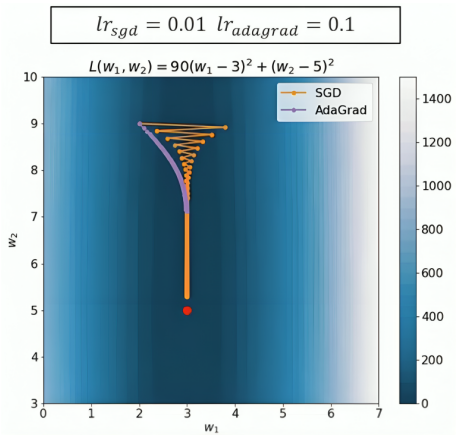


Figure 1.5: SGD vs AdaGrad. AdaGrad stops before getting close to the minimum.

1.4.2 RMSProp

RMSProp Modified version of AdaGrad that down-weighs the gradient history $s^{(t)}$: RMSProp

$$\mathbf{s}^{(t+1)} = \beta\mathbf{s}^{(t)} + (1-\beta)\nabla\mathcal{L}(\boldsymbol{\theta}^{(t)}) \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$
$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\texttt{lr}}{\sqrt{\mathbf{s}^{(t+1)}} + \varepsilon} \odot \nabla\mathcal{L}(\boldsymbol{\theta}^{(t)})$$

where $\beta \in [0, 1]$ (typically 0.9 or higher) makes $s^{(t)}$ an exponential moving average.

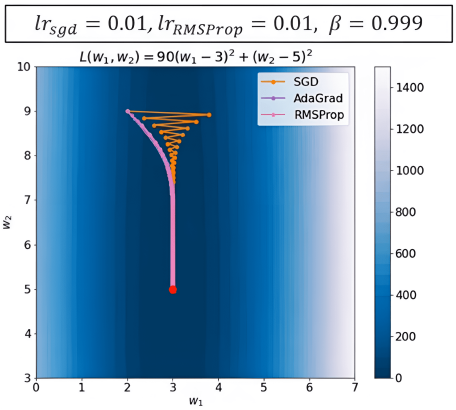


Figure 1.6: SGD vs AdaGrad vs RMSProp