Machine learning

David Zarzoso
Aix Marseille Univ, CNRS, Centrale Med, M2P2 UMR 7340, Marseille

Gradient descent methods

Convexity of least-square linear regression (on the board)

Gradient descent variants

- Momentum
- Nesterov accelerated gradient
- Adagrad
- RMSprop
- Adam

Bibliography

- *Neural Networks and Deep Learning,* Charu C. Aggarwal
- Some other open-access papers that are in the slides

Exercise

Demonstrate that the least-square linear regression is a convex problem

Solution: on the board

Problem of the *standard* gradient descent

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

Learning rate

η small → the algorithm converges slowly to the solution
η large → the algorithm jumps around the solution or even diverges

Stochastic Gradient Descent (SGD)

Mini-batch Gradient Descent

Full-batch Gradient Descent (GD)

A good compromise: batch size =256-512

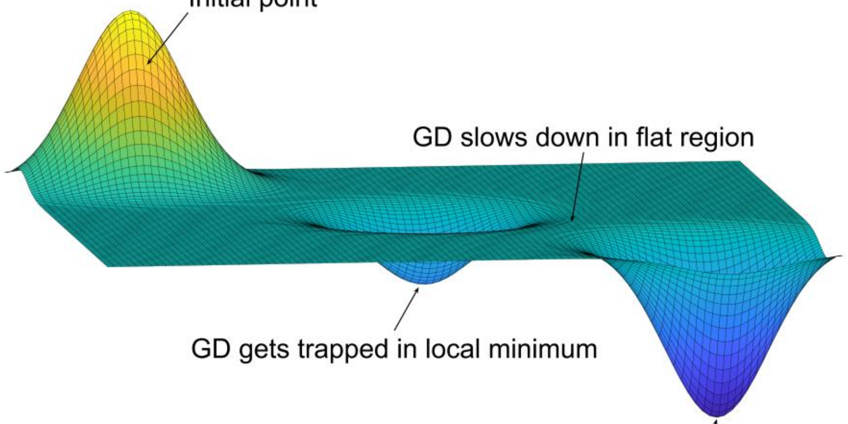
Problem of the *standard* gradient descent

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

Learning rate

η small → the algorithm converges slowly to the solution
η large → the algorithm jumps around the solution or even diverges

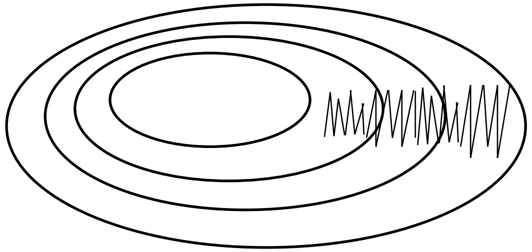
The updated parameters depend only on the learning rate and the gradient at that moment
No past history is taken into account
It can slow down in saddle points, where the gradient is ≈0 and get stuck in local minima



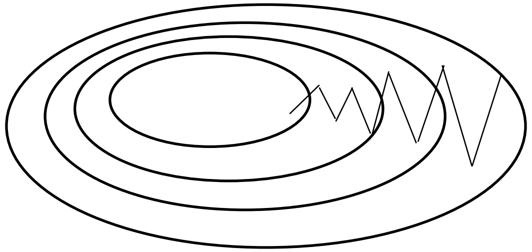
Two main ideas

1. Include past history of the gradient (momentum)
2. Adapt the learning rate (adaptive gradient methods)

Nesterov accelerated gradient (NAG) descent



Without momentum



With momentum

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

$$\boldsymbol{v}_n = \gamma \boldsymbol{v}_{n-1} + \eta \left.\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right|_{\boldsymbol{\theta}_n - \gamma \boldsymbol{v}_{n-1}}, \text{ with } 0 \leq \gamma < 1$$

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \boldsymbol{v}_n$$

In $\boldsymbol{v}_n$ we have the present gradient and the previous ones encapsulated in $\boldsymbol{v}_{n-1}$
Usually $\gamma \approx 0.9 - 0.95$
What's new? Now we evaluate the gradient at $\boldsymbol{\theta}_n - \gamma \boldsymbol{v}_{n-1}$, which is an approximation of the new position after the upgrade → **Idea of anticipation**

Adaptive Gradient (AdaGrad) descent

https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf

Adapt the learning rate for each component of $\boldsymbol{\theta}$ based on the past history of the gradient for that component
Idea behind: some parameters might need more frequent updates than others

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

$$\downarrow$$

$$\theta_{j,n+1} = \theta_{j,n} - \eta \frac{\partial \mathcal{L}}{\partial \theta_j}, \text{ for each } j$$

Contains all the previous squared gradients

$$G_{j,n} = G_{j,n-1} + \left(\left. \frac{\partial \mathcal{L}}{\partial \theta_j} \right|_{\boldsymbol{\theta}_n} \right)^2$$

$$\theta_{j,n+1} = \theta_{j,n} - \frac{\eta}{\sqrt{G_{j,n}} + \varepsilon} \frac{\partial \mathcal{L}}{\partial \theta_j}$$

Small constant (10^{-8})to avoid division by zero

$G_{j,n}$ measures the historical magnitude of the gradient, rather than its sign, encouraging evolution along gently sloping directions with consistent sign of the gradient
☺ There is no need to tune η anymore, since the effective learning rate decreases with the number of iterations until the gradient vanishes
☹ However, the effective learning rate can decrease too fast and AdaGrad will stop minimizing the loss function

Root Mean Square Propagation (RMSProp)

Instead of adding the squared gradients, we use exponential averaging

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

$$\downarrow$$

$$\theta_{j,n+1} = \theta_{j,n} - \eta \frac{\partial \mathcal{L}}{\partial \theta_j}, \text{ for each } j$$

Contains all the previous squared gradients

$$G_{j,n} = (1-\gamma) G_{j,n-1} + \gamma \left(\left. \frac{\partial \mathcal{L}}{\partial \theta_j} \right|_{\boldsymbol{\theta}_n} \right)^2 \quad \text{, with } 0 \leq \gamma < 1$$

$$\theta_{j,n+1} = \theta_{j,n} - \frac{\eta}{\sqrt{G_{j,n}} + \varepsilon} \frac{\partial \mathcal{L}}{\partial \theta_j}$$

Small constant (10^{-8})to avoid division by zero

☺ The importance of past gradients decays exponentially with the number of iterations → The learning process is not slowed down prematurely

AdaDelta (https://arxiv.org/abs/1212.5701)

Have a look also at https://arxiv.org/pdf/1206.1106

Similar update as RMSProp, but it eliminates the need for a global learning rate
The global learning rate is actually computed as exponential average of incremental updates in previous iterations

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

$$\downarrow$$

$$\theta_{j,n+1} = \theta_{j,n} - \eta \frac{\partial \mathcal{L}}{\partial \theta_j}, \text{ for each } j$$

Contains all the previous squared gradients

$$G_{j,n} = (1-\gamma) G_{j,n-1} + \gamma \left(\left. \frac{\partial \mathcal{L}}{\partial \theta_j} \right|_{\boldsymbol{\theta}_n} \right)^2 \quad \text{, with } 0 \leq \gamma < 1$$

$$\delta_{j,n} = (1-\gamma) \delta_{j,n-1} + \gamma \left(\frac{\sqrt{\delta_{j,n-1}} + \varepsilon}{\sqrt{G_{j,n-1}} + \varepsilon} \left. \frac{\partial \mathcal{L}}{\partial \theta_j} \right|_{\boldsymbol{\theta}_n} \right)^2$$

$$\theta_{j,n+1} = \theta_{j,n} - \frac{\sqrt{\delta_{j,n}} + \varepsilon}{\sqrt{G_{j,n}} + \varepsilon} \frac{\partial \mathcal{L}}{\partial \theta_j}$$

Small constant (10^{-8})to avoid division by zero

Adam (Adaptive momentum, https://arxiv.org/pdf/1412.6980)

The most popular one and probably the most used for machine learning

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$$

$$\downarrow$$

$$\theta_{j,n+1} = \theta_{j,n} - \eta \frac{\partial \mathcal{L}}{\partial \theta_j}, \text{ for each } j$$

w.r.t. RMSprop, two main differences:

1. The gradient is replaced with its exponentially smoothed value in order to incorporate momentum
2. The learning rate depends on the iteration and is adjusted to account for unrealistic initialization of two exponential smoothing

Contains all the previous squared gradients

$$m_{j,n} = \beta_1 m_{j,n-1} + (1-\beta_1) \left.\frac{\partial \mathcal{L}}{\partial \theta_j}\right|_{\boldsymbol{\theta}_n}$$

Initialized to 0

$$v_{j,n} = \beta_2 v_{j,n-1} + (1-\beta_2) \left(\left.\frac{\partial \mathcal{L}}{\partial \theta_j}\right|_{\boldsymbol{\theta}_n}\right)^2$$

Initialized to 0

$$\hat{m}_{j,n} = \frac{m_{j,n}}{1-\beta_1^n} \qquad \hat{v}_{j,n} = \frac{v_{j,n}}{1-\beta_2^n}$$

$$\theta_{j,n+1} = \theta_{j,n} - \frac{\eta}{\sqrt{\hat{v}_{j,n}} + \varepsilon} \hat{m}_{j,n}$$

$\eta = 0.001,\ \beta_1 = 0.9$ and $\beta_2 = 0.999$