ENM531: Data-driven modeling and probabilistic scientific computing

Lecture #6: Optimization

Paris Perdikaris
February 5, 2019

Linear regression: Things we didn't cover

- Robust linear regression: Laplace/Student-T likelihoods
- Sparse regression: Laplace prior on model parameters (=> L1 penalty)
- Full Bayesian inference
- The expected square loss and the bias-variance decomposition

Reading

- Linear regression and Bayesian linear regression:
  - Chapter 7 (Murphy, 2012)
  - Chapter 3 (Bishop, 2006)

1. Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
2. Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Objectives

At its core, machine learning is all about integration (e.g., computing expectations, etc.) and ***optimization***. Today we'll revisit some basic concepts in optimization, and introduce them in the context of training machine learning algorithms.

Specifically, we'll cover:

- The definition of gradients and Hessians.
- The gradient descent algorithm.
- Newton's algorithm.
- Applications to linear regression.
- Stochastic gradient descent.
- Modern variants of stochastic gradient descent.

Objectives

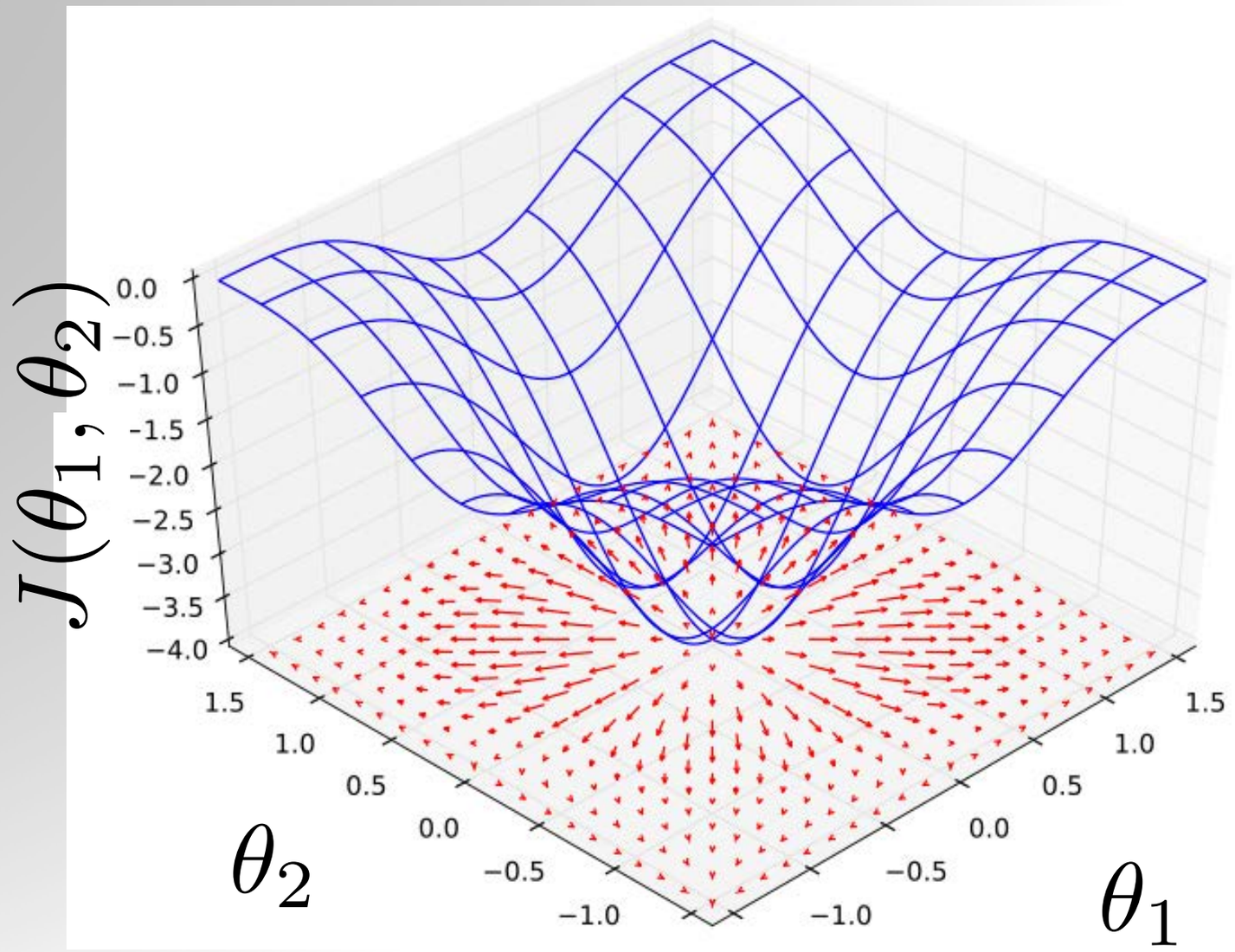
At its core, machine learning is all about integration (e.g., computing expectations, etc.) and ***optimization***. Today we'll revisit some basic concepts in optimization, and introduce them in the context of training machine learning algorithms.

Specifically, we'll cover:

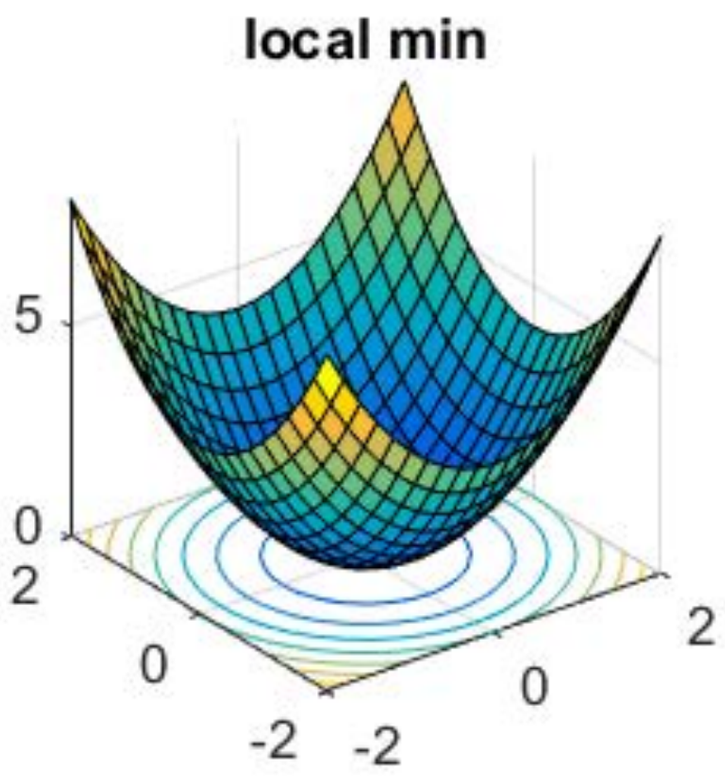
- The definition of gradients and Hessians.
- The gradient descent algorithm.
- Newton's algorithm.
- Applications to linear regression.
- Stochastic gradient descent.
- Modern variants of stochastic gradient descent.

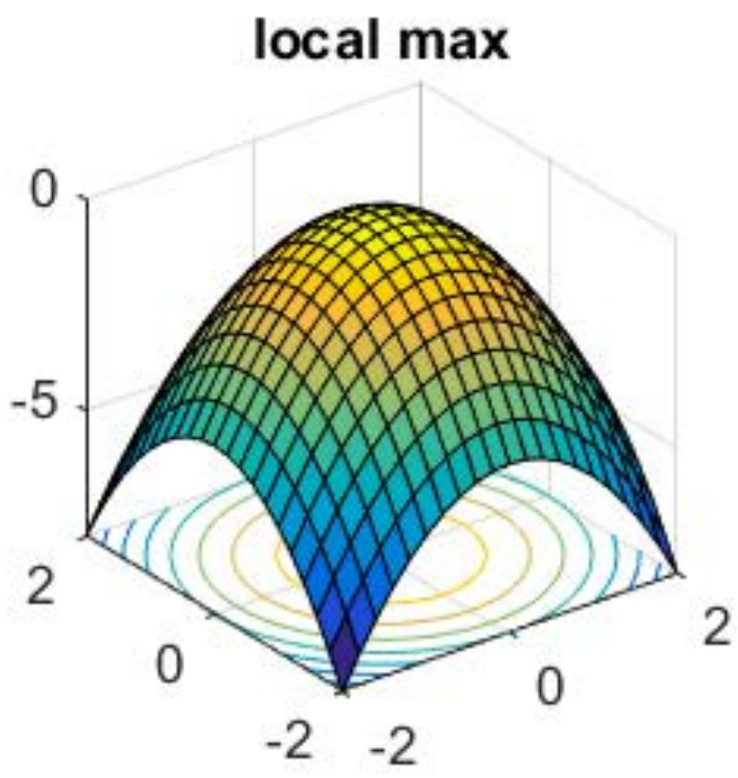
Gradients

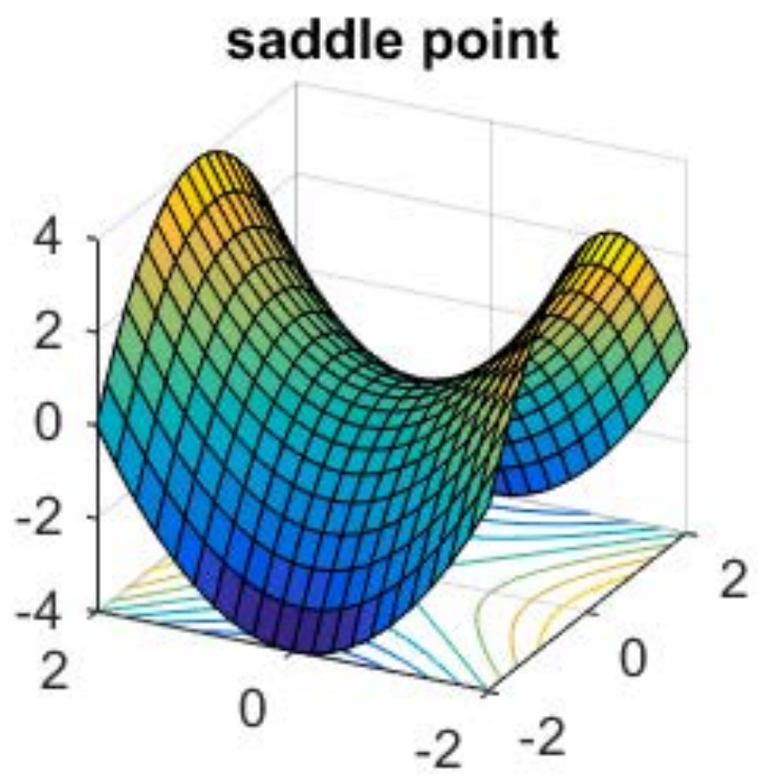
$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_1} \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{\theta})}{\partial \theta_n} \end{bmatrix}$$



Minima, maxima, and saddle points



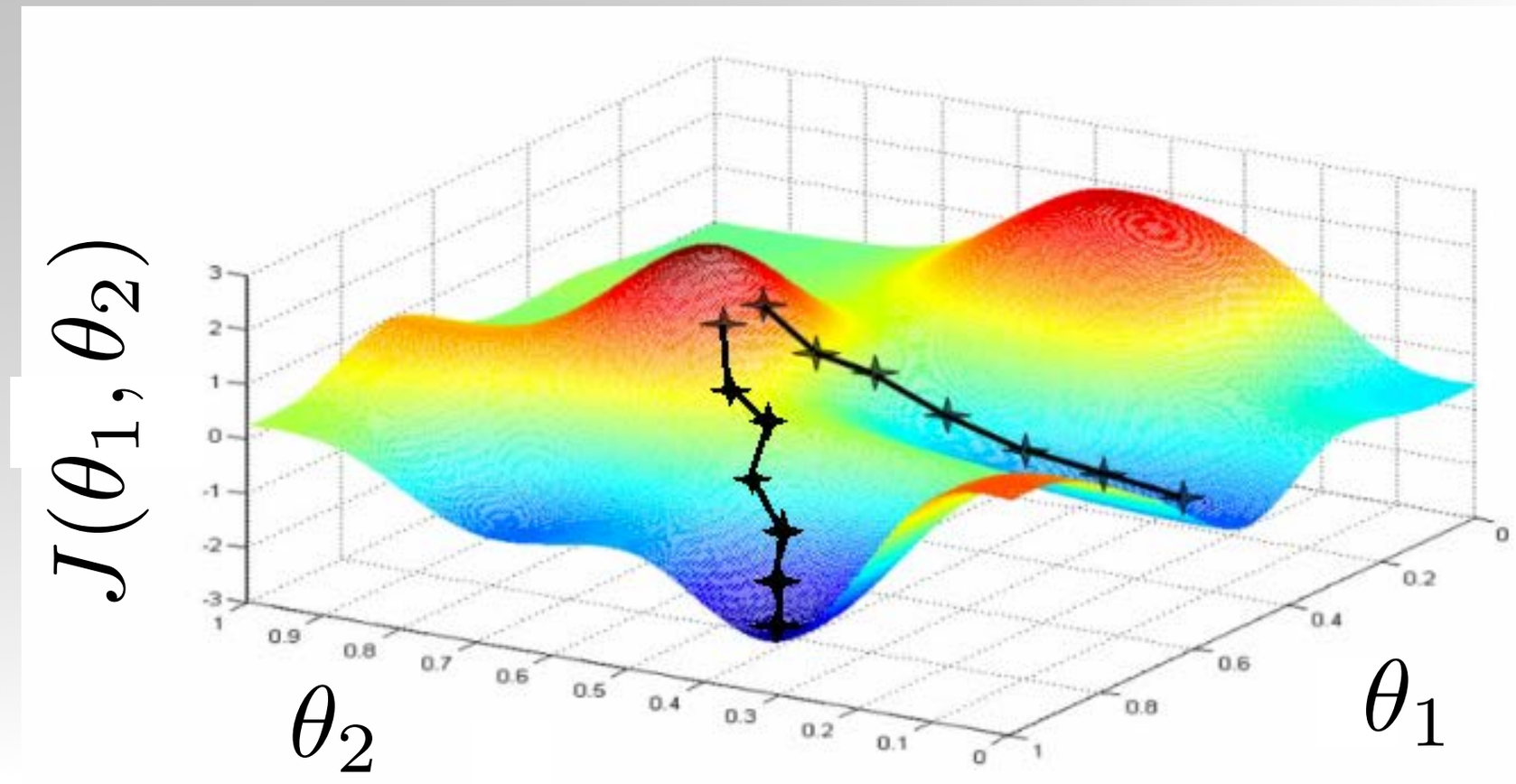




Gradient descent

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} J(\boldsymbol{\theta})$$

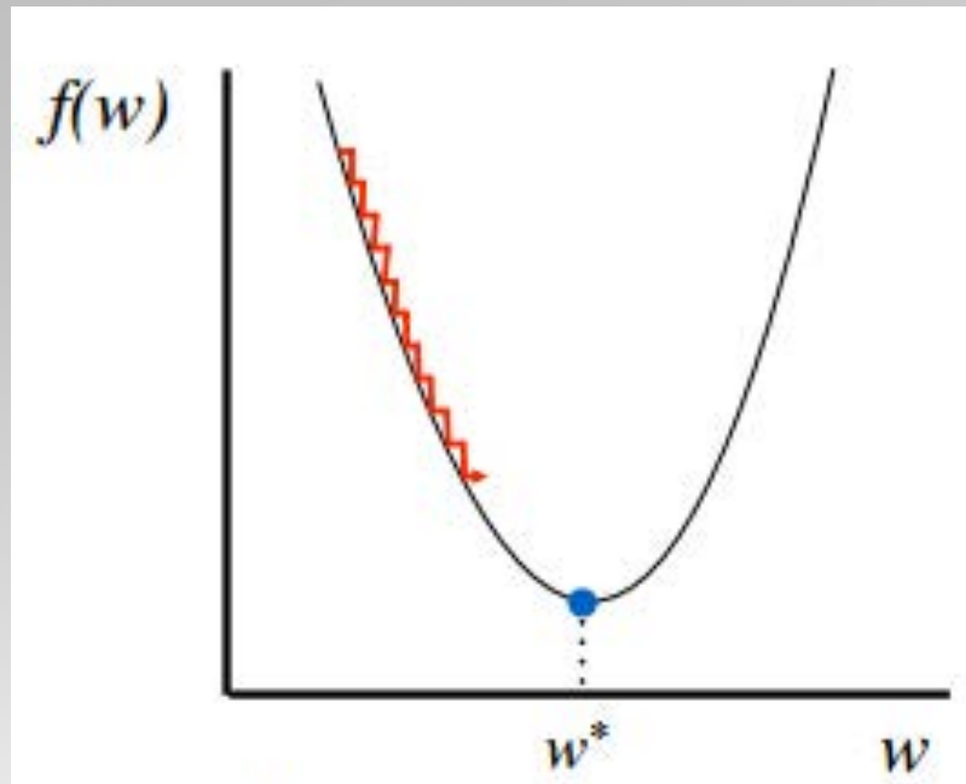
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$



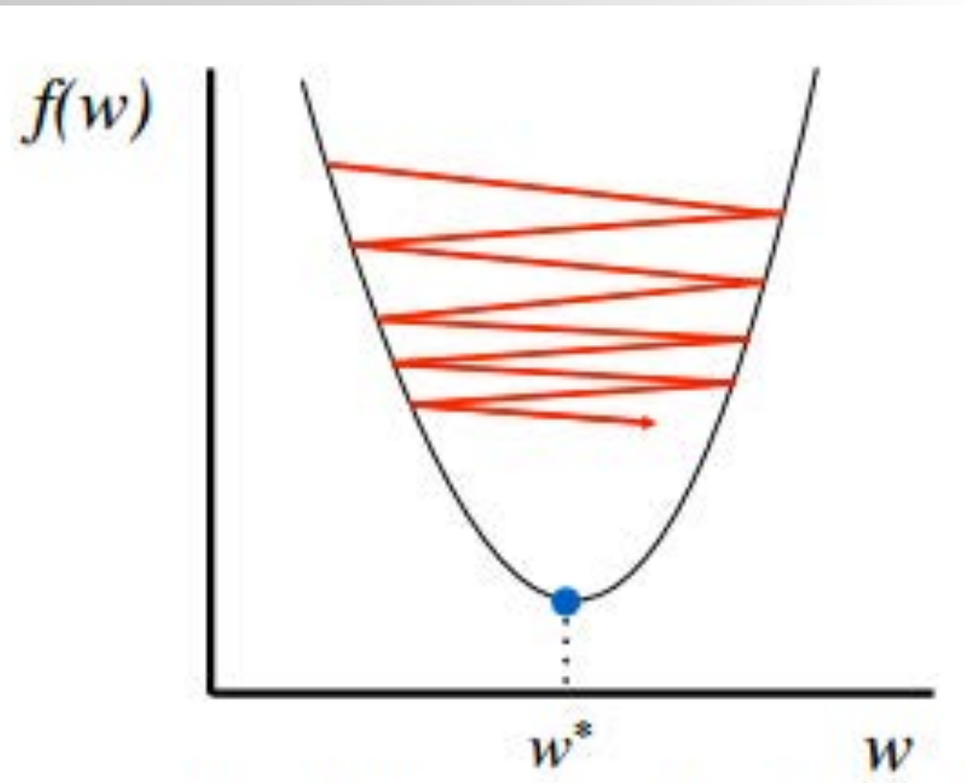
Gradient descent

$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

Effect of the learning rate



Too small: converge very slowly

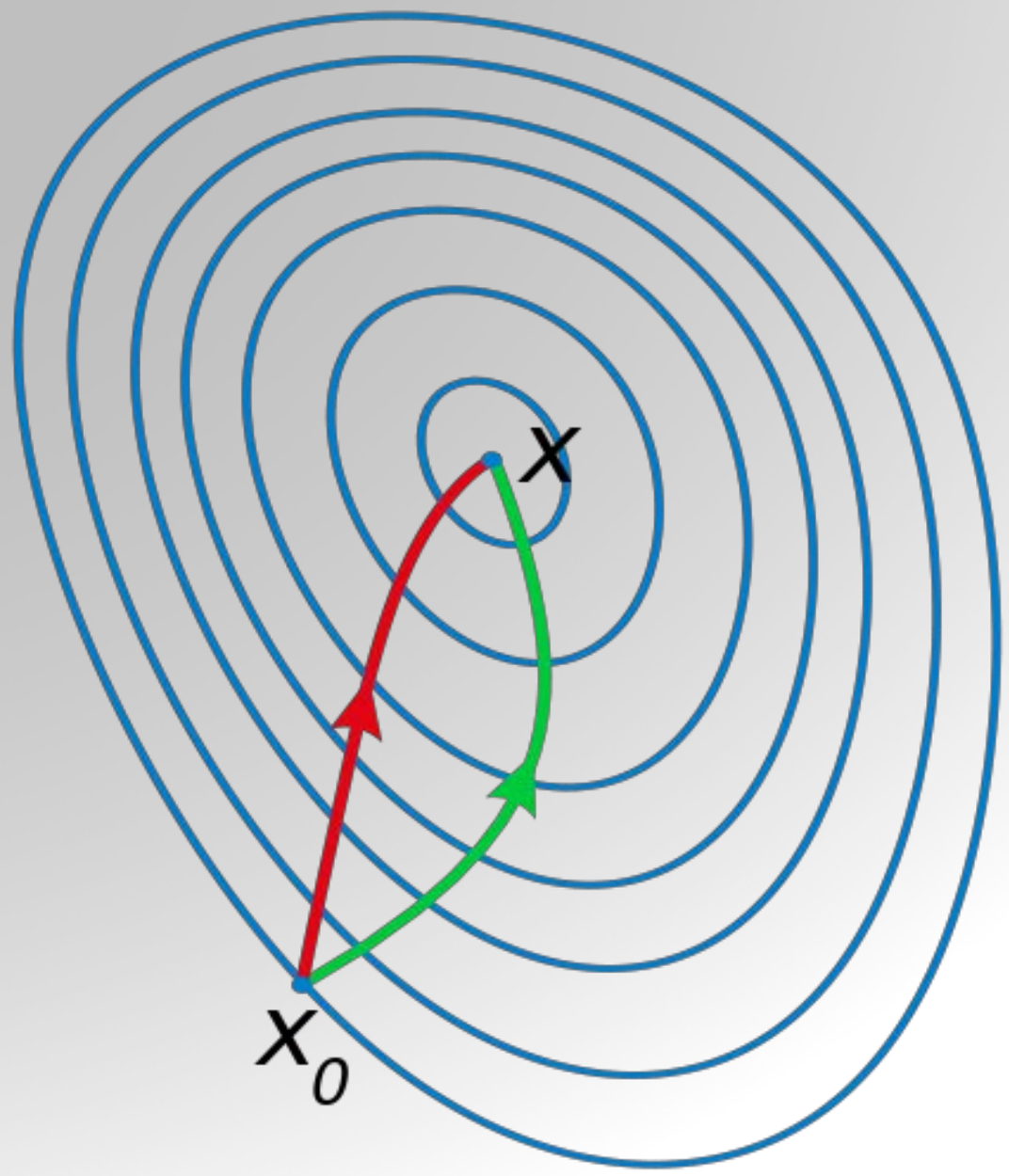


Too big: overshoot and even diverge

Hessian

$$\nabla^2_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1^2} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2^2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_2 \partial \theta_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d \partial \theta_1} & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d \partial \theta_2} & \cdots & \frac{\partial^2 f(\boldsymbol{\theta})}{\partial \theta_d^2} \end{bmatrix}$$

Gradient descent vs Newton's method



BFGS

Broyden–Fletcher–Goldfarb–Shanno algorithm

From Wikipedia, the free encyclopedia

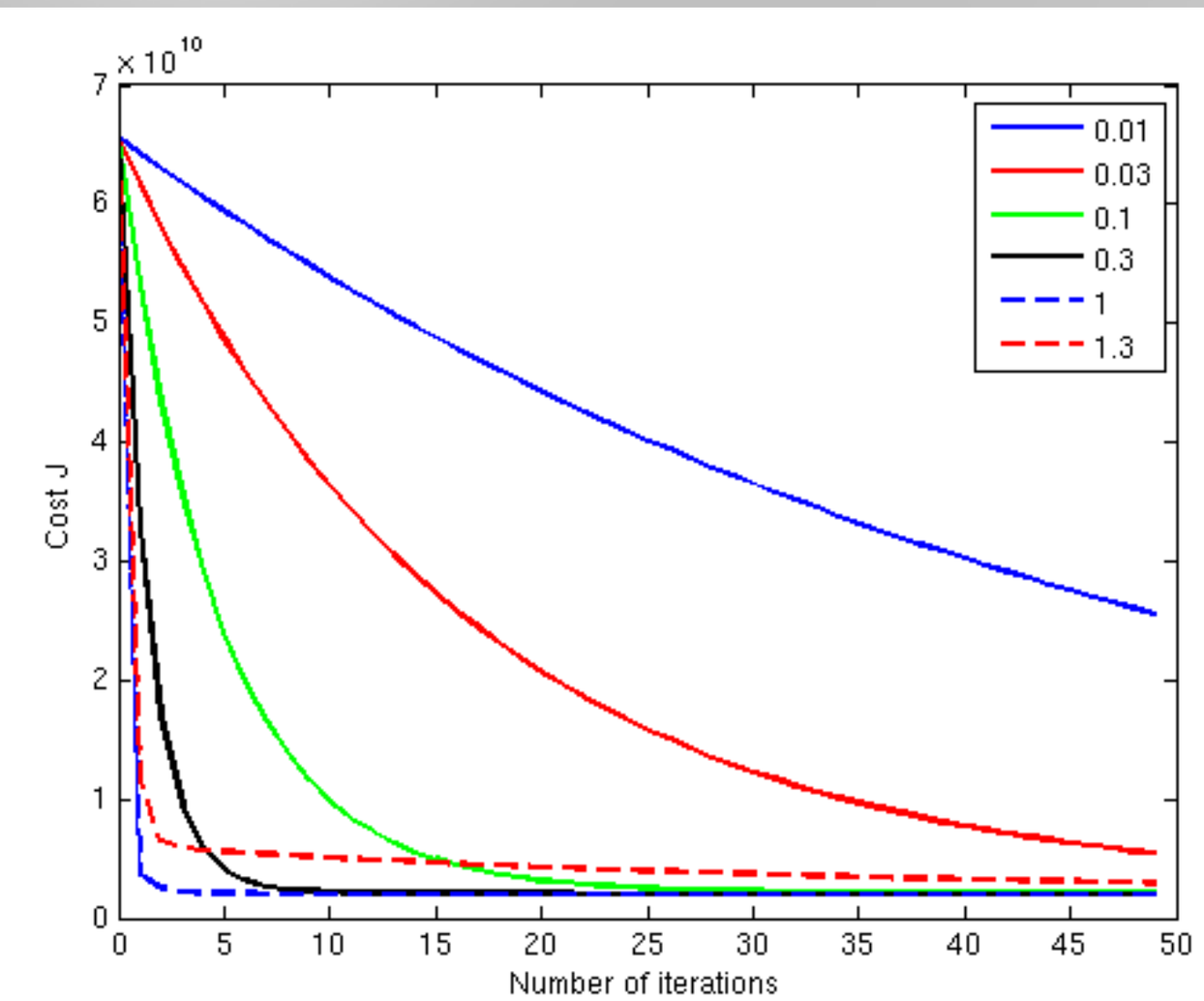
In numerical optimization, the **Broyden–Fletcher–Goldfarb–Shanno** (**BFGS**) **algorithm** is an iterative method for solving unconstrained nonlinear optimization problems.[1]

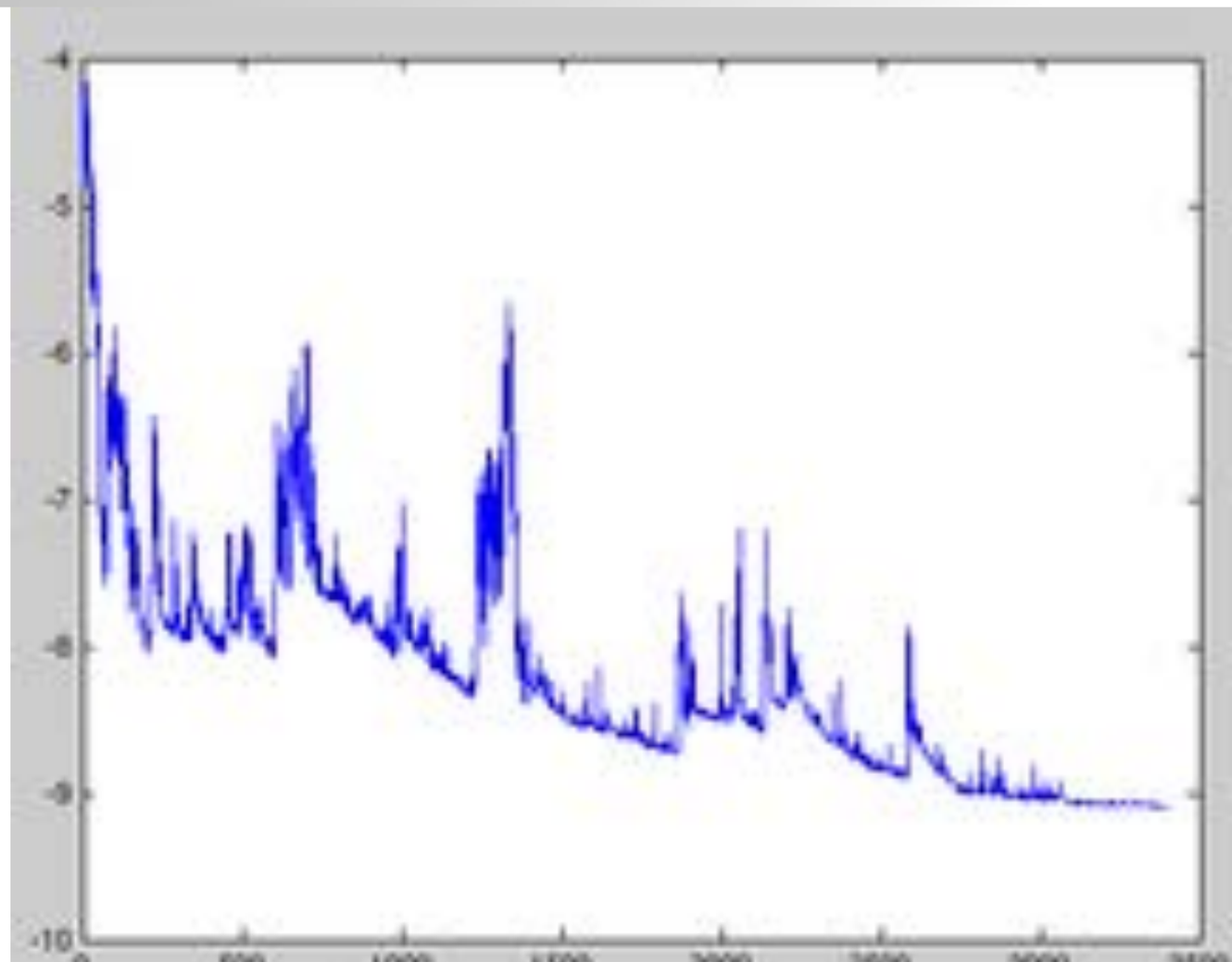
The BFGS method belongs to quasi-Newton methods, a class of hill-climbing optimization techniques that seek a stationary point of a (preferably twice continuously differentiable) function. For such problems, a necessary condition for optimality is that the gradient be zero. Newton's method and the BFGS methods are not guaranteed to converge unless the function has a quadratic Taylor expansion near an optimum. However, BFGS has proven to have good performance even for non-smooth optimizations.[2]

In quasi-Newton methods, the Hessian matrix of second derivatives doesn't need to be evaluated directly. Instead, the Hessian matrix is approximated using updates specified by gradient evaluations (or approximate gradient evaluations). Quasi-Newton methods are generalizations of the secant method to find the root of the first derivative for multidimensional problems. In multi-dimensional problems, the secant equation does not specify a unique solution, and quasi-Newton methods differ in how they constrain the solution. The BFGS method is one of the most popular members of this class.[3] Also in common use is L-BFGS, which is a limited-memory version of BFGS that is particularly suited to problems with very large numbers of variables (e.g., >1000). The BFGS-B[4] variant handles simple box constraints.

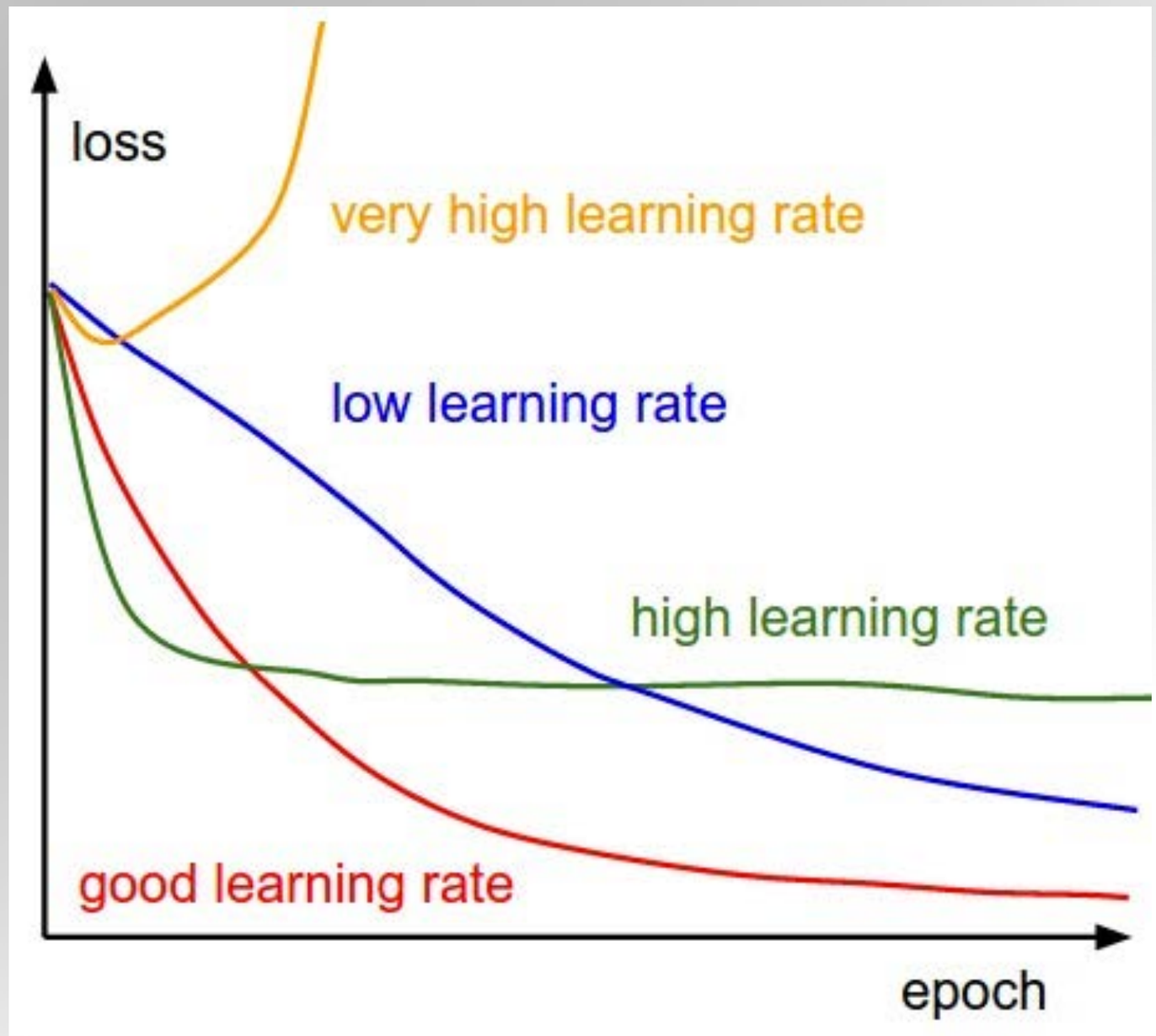
The algorithm is named after Charles George Broyden, Roger Fletcher, Donald Goldfarb and David Shanno.

Gradient descent vs SGD





Gradient descent vs SGD



Remarks

Vanilla mini-batch gradient descent, however, does not guarantee good convergence, but offers a few challenges that need to be addressed:

- Choosing a proper learning rate can be difficult.

- Learning rate schedules (i.e., adjusting the learning rate during training) has to be defined in advance and it is thus unable to adapt to a dataset's characteristics.

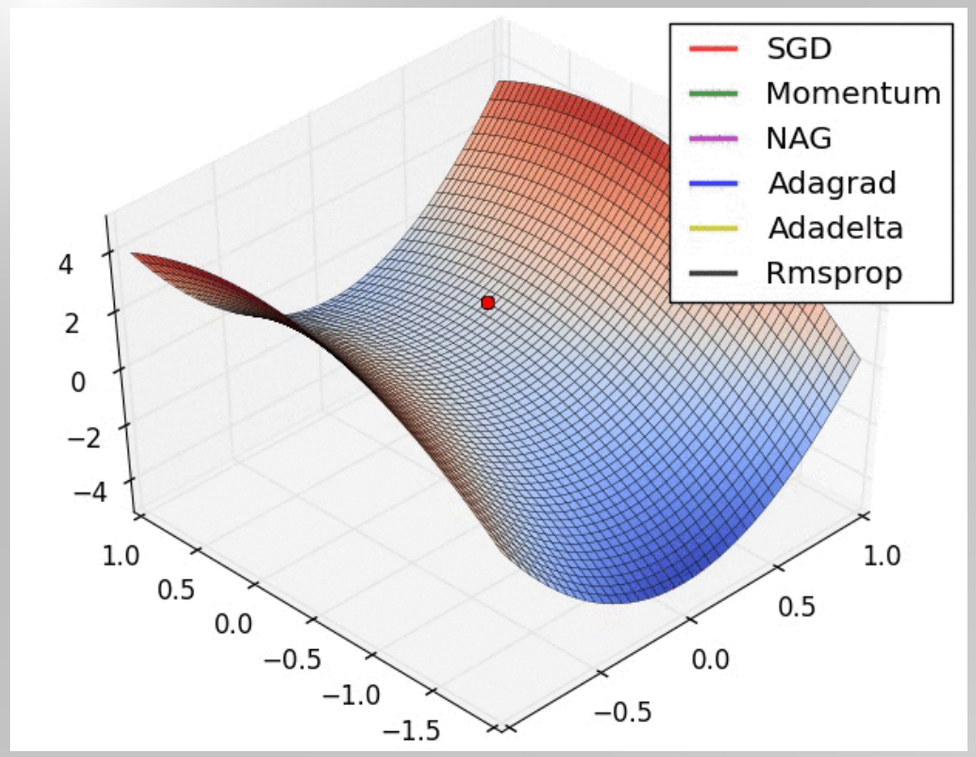
- The same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.

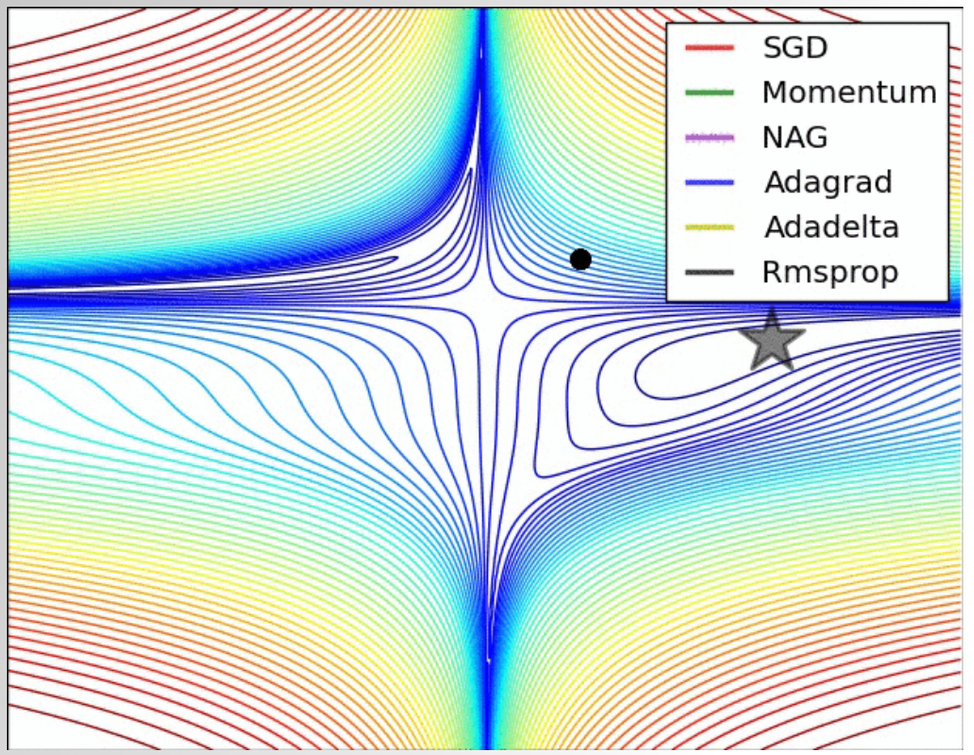
- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima. Dauphin et al. argue that the difficulty arises in fact not from local minima but from saddle points, i.e. points where one dimension slopes up and another slopes down. These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Dauphin, Y., et. al. (2014). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. arXiv, 1–14.

Modern SGD variants

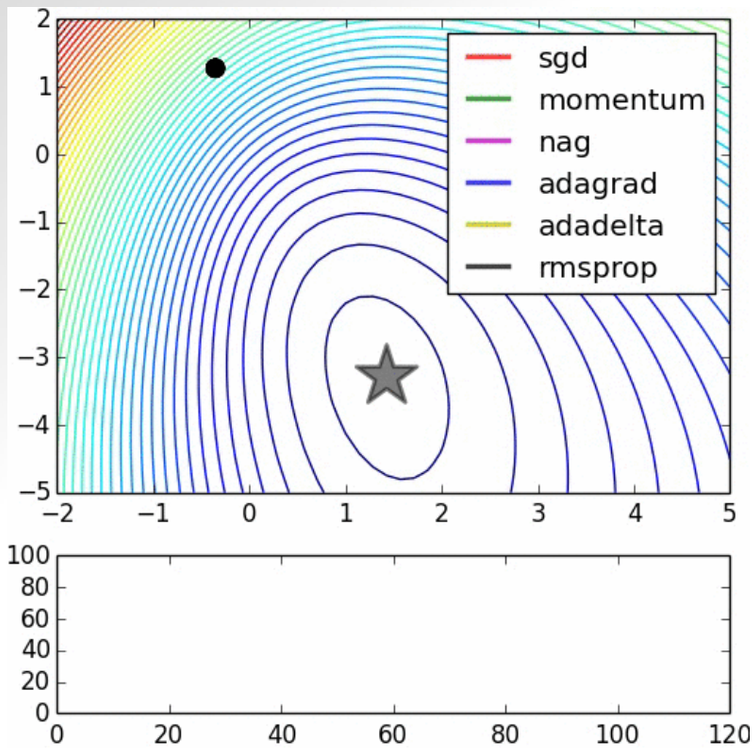




http://ruder.io/optimizing-gradient-descent/

https://distill.pub/2017/momentum/

http://louistiao.me/notes/visualizing-and-animating-optimization-algorithms-with-matplotlib/



*animation credit: Alec Redford