ENM 531: Data-driven Modeling and Probabilistic Scientific Computing

Lecture #8: Variational inference

Paris Perdikaris
February 10, 2021

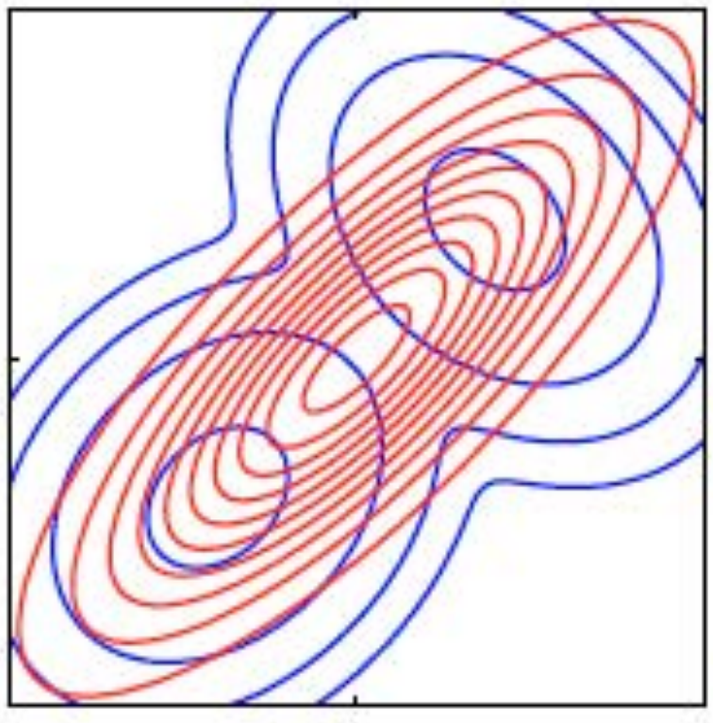
Variational inference

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta|)p(\theta)}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\theta|)p(\theta)}{\int p(\mathcal{D}|\theta|)p(\theta)d\theta}$$

- Variational inference provides a computational framework for approximate Bayesian inference.
- The idea is that we'll approximate the posterior distribution with a family of distributions that is easy to work with.
- It will provide us with a set of tools for transforming the sampling problem (integration) to an optimization problem, that can be scaled to large models (i.e. with many parameters) and large data-sets.
- It also tends to favor approximations that underestimate the variance, and it usually will result in approximate distributions that get the means right but underestimate the variance.

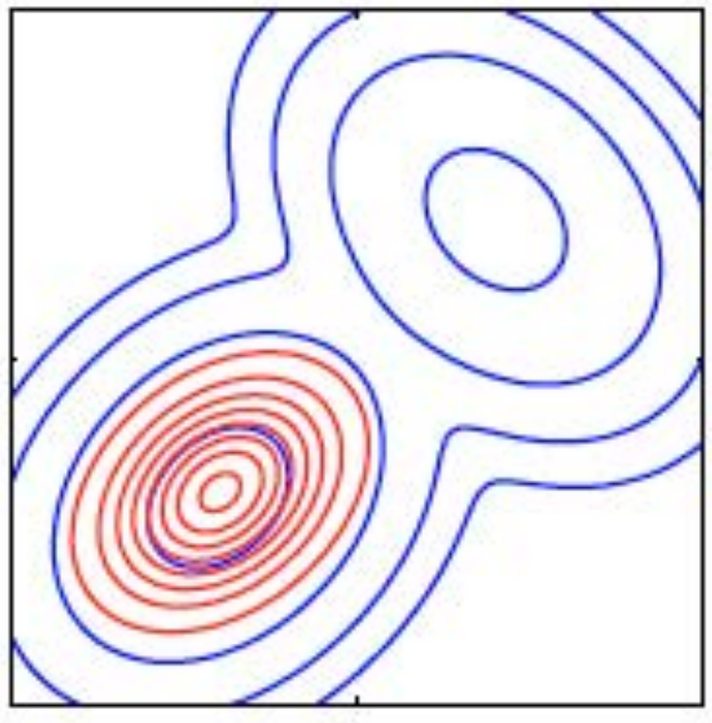
Variational inference

$\mathbb{KL}[q_\phi(x)||p(x)]$



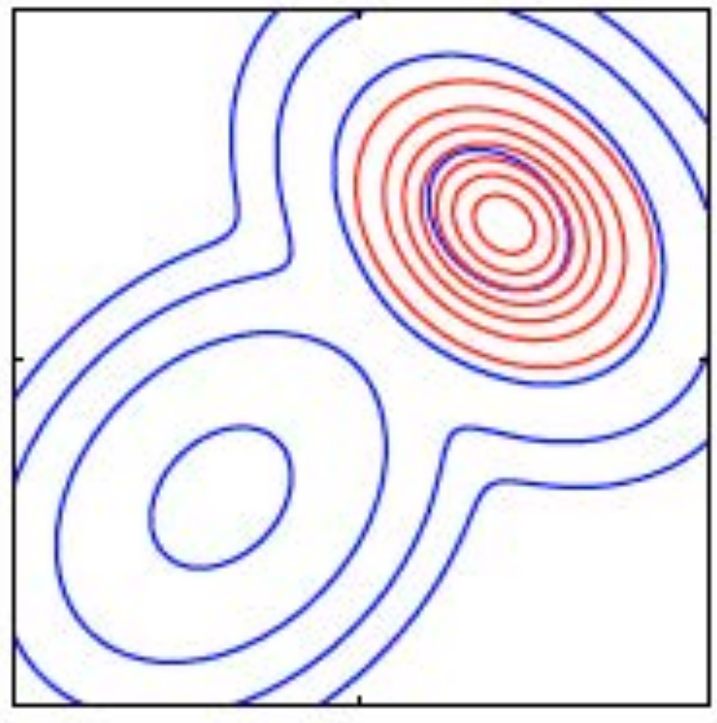
KL

$\mathbb{KL}[p(x)||q_\phi(x)]$



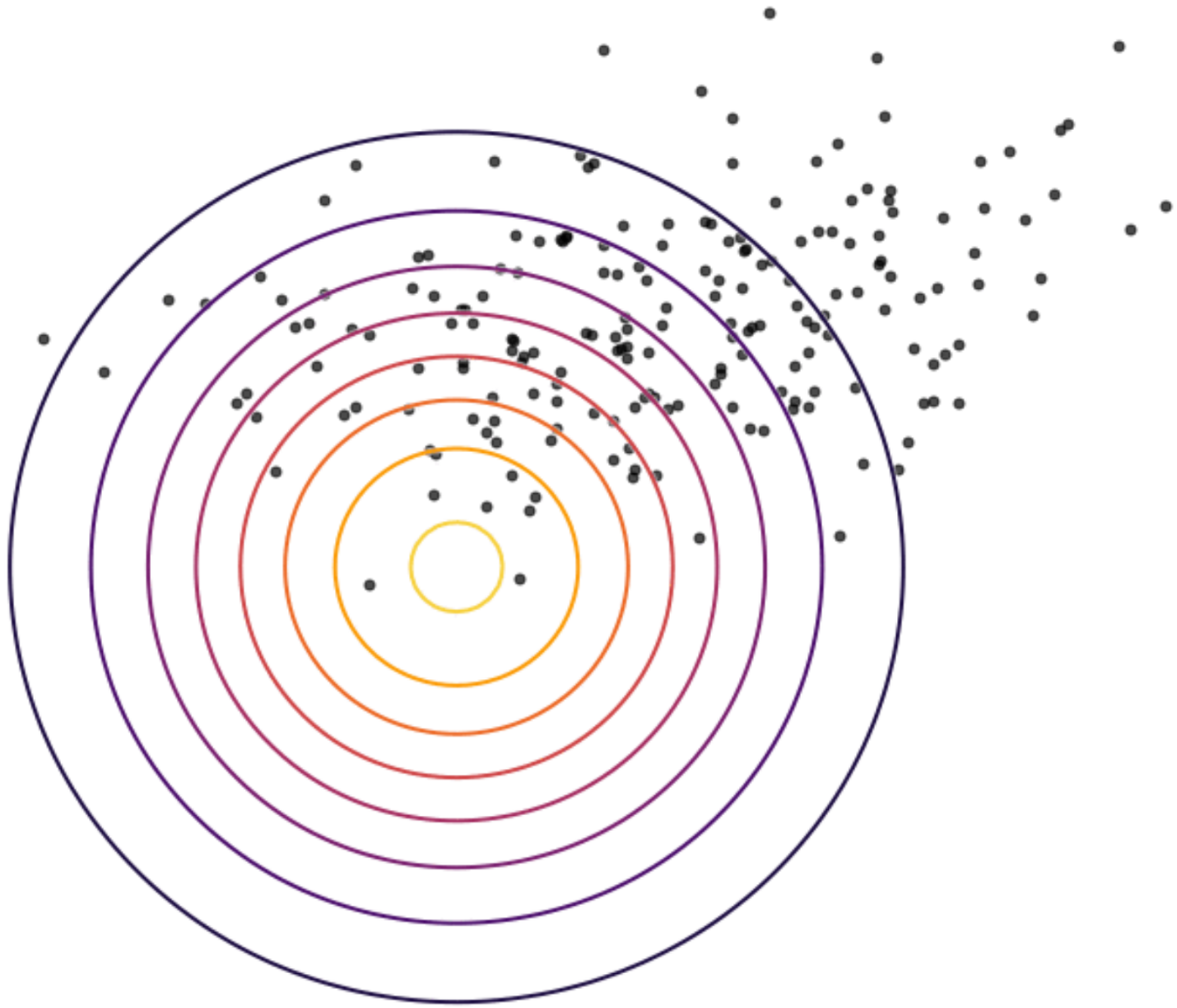
Reverse KL

$\mathbb{KL}[p(x)||q_\phi(x)]$



Reverse KL

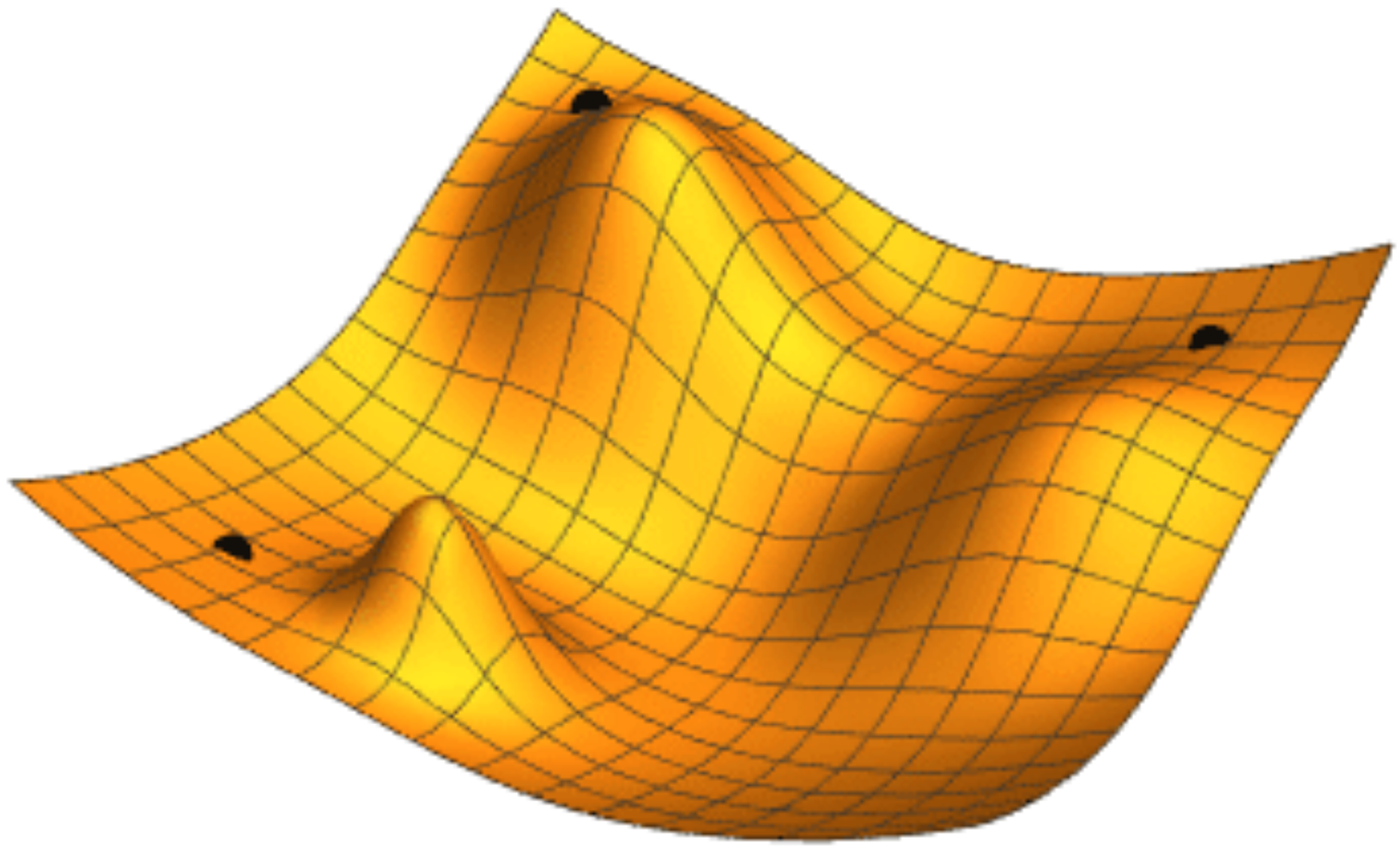
Variational inference



Gradient descent

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

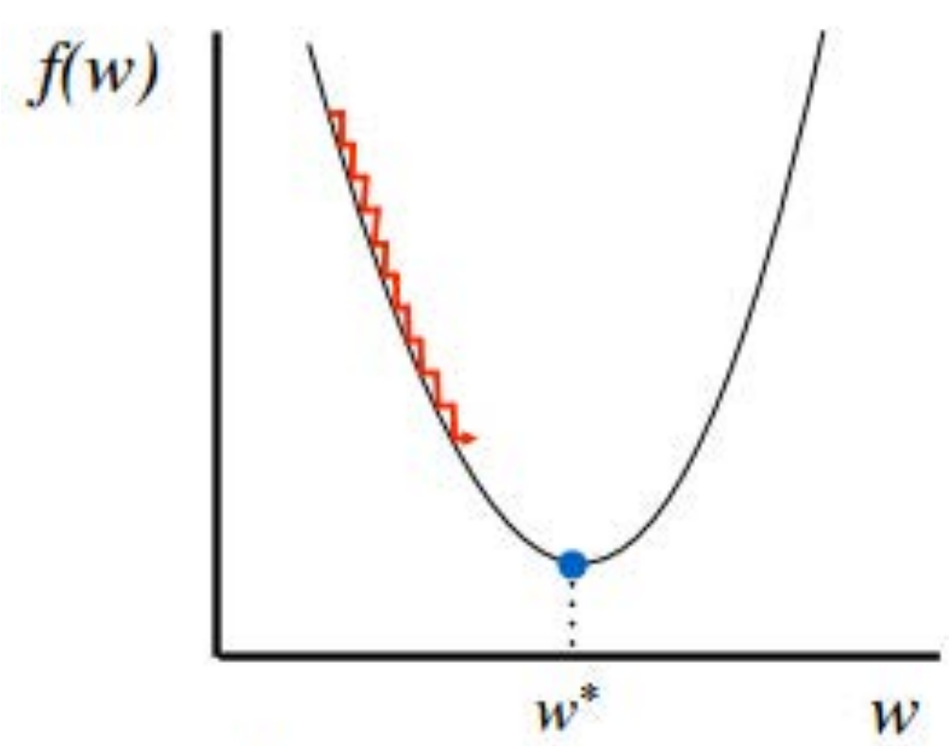
$$\boldsymbol{\theta}_{n+1} = \boldsymbol{\theta}_n - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$



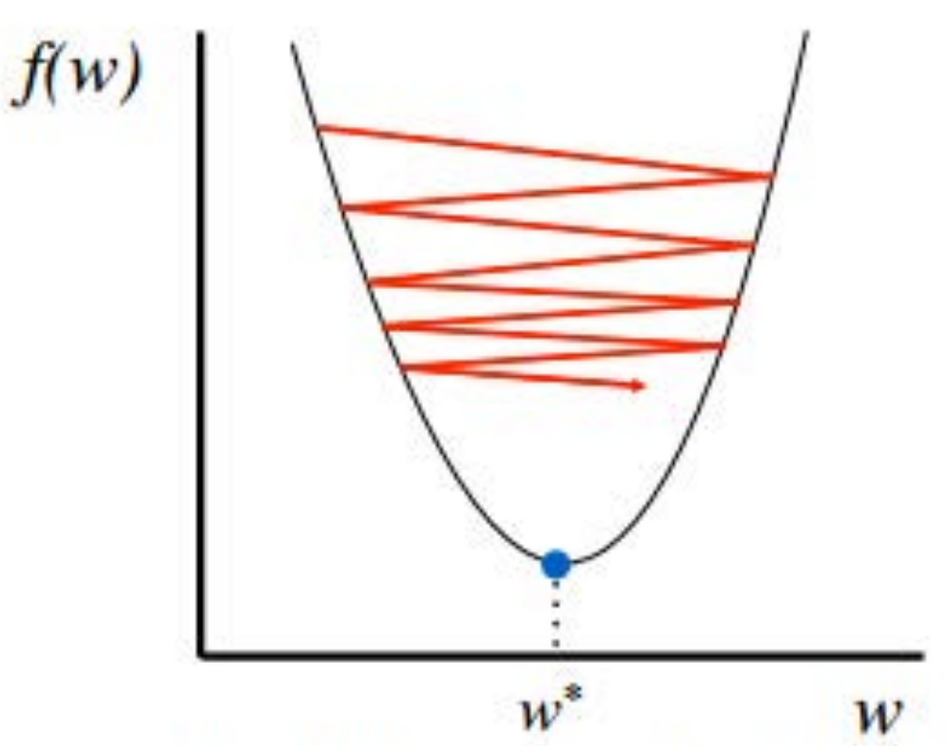
Gradient descent

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

Effect of the learning rate



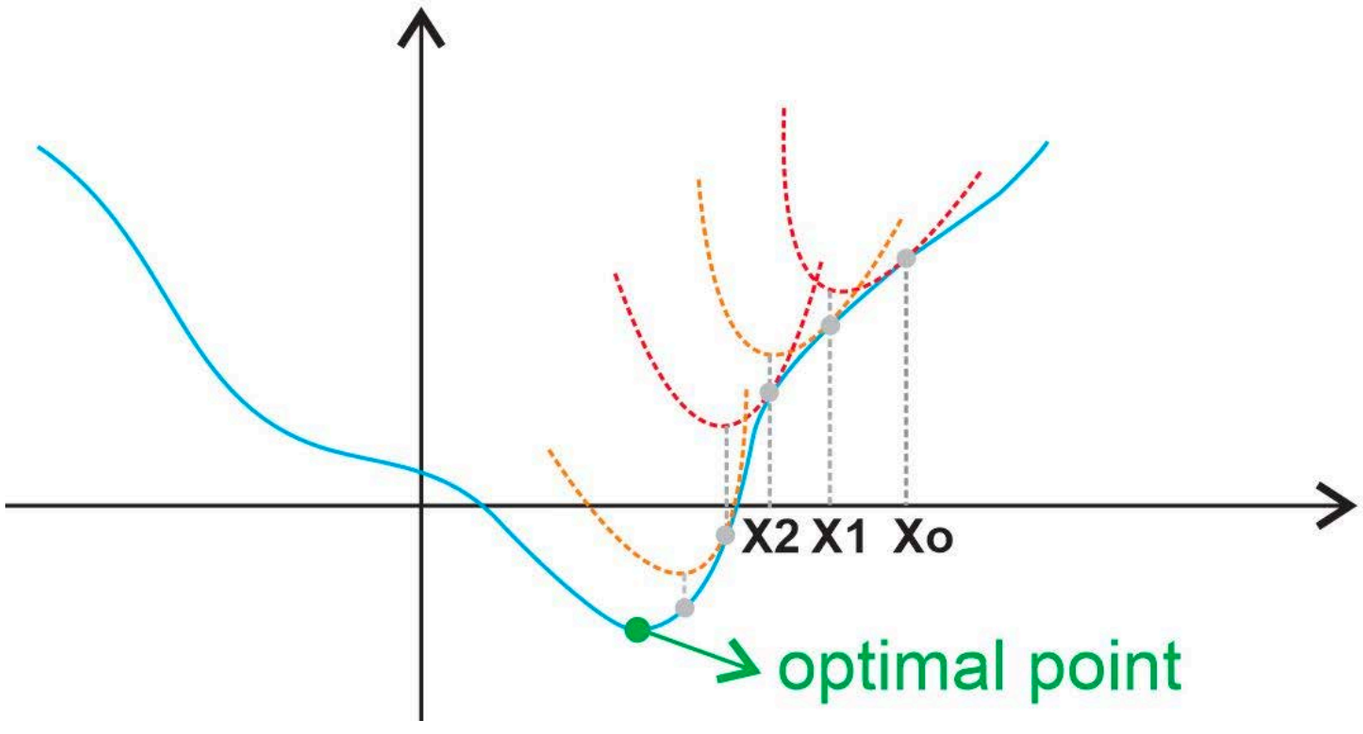
Too small: converge very slowly



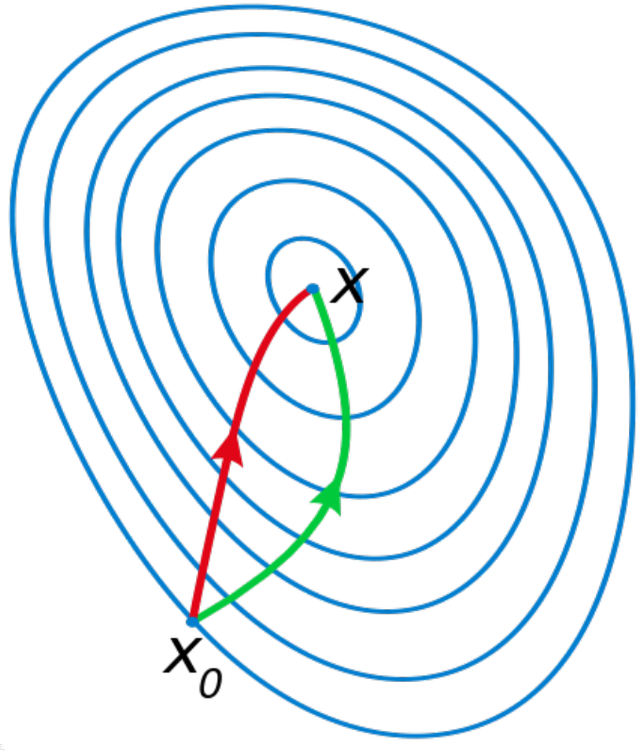
Too big: overshoot and even diverge

Hessian

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



Local quadratic approximation
of the loss == Newton's method



Gradient descent vs Newton

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]

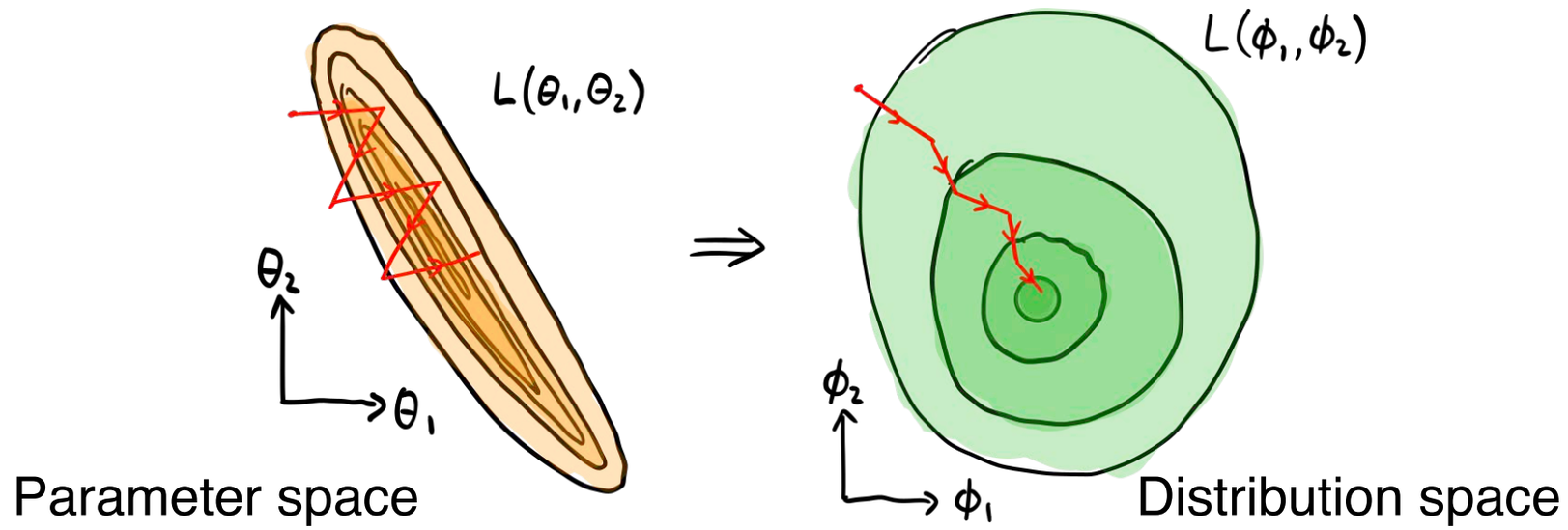
The BFGS method belongs to quasi-Newton methods, a class of hill-climbing optimization techniques that seek a stationary point of a (preferably twice continuously differentiable) function. For such problems, a necessary condition for optimality is that the gradient be zero. Newton's method and the BFGS 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 natural gradient descent

Motivation: If our objective is to minimize the loss function (maximizing the likelihood), then it is natural that we taking step in the space of all possible likelihoods, realizable by the parameters θ. As the likelihood function itself is a probability distribution, we call this space distribution space. Thus it makes sense to take the steepest descent direction in this distribution space instead of parameter space.



Parameter space Distribution space

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

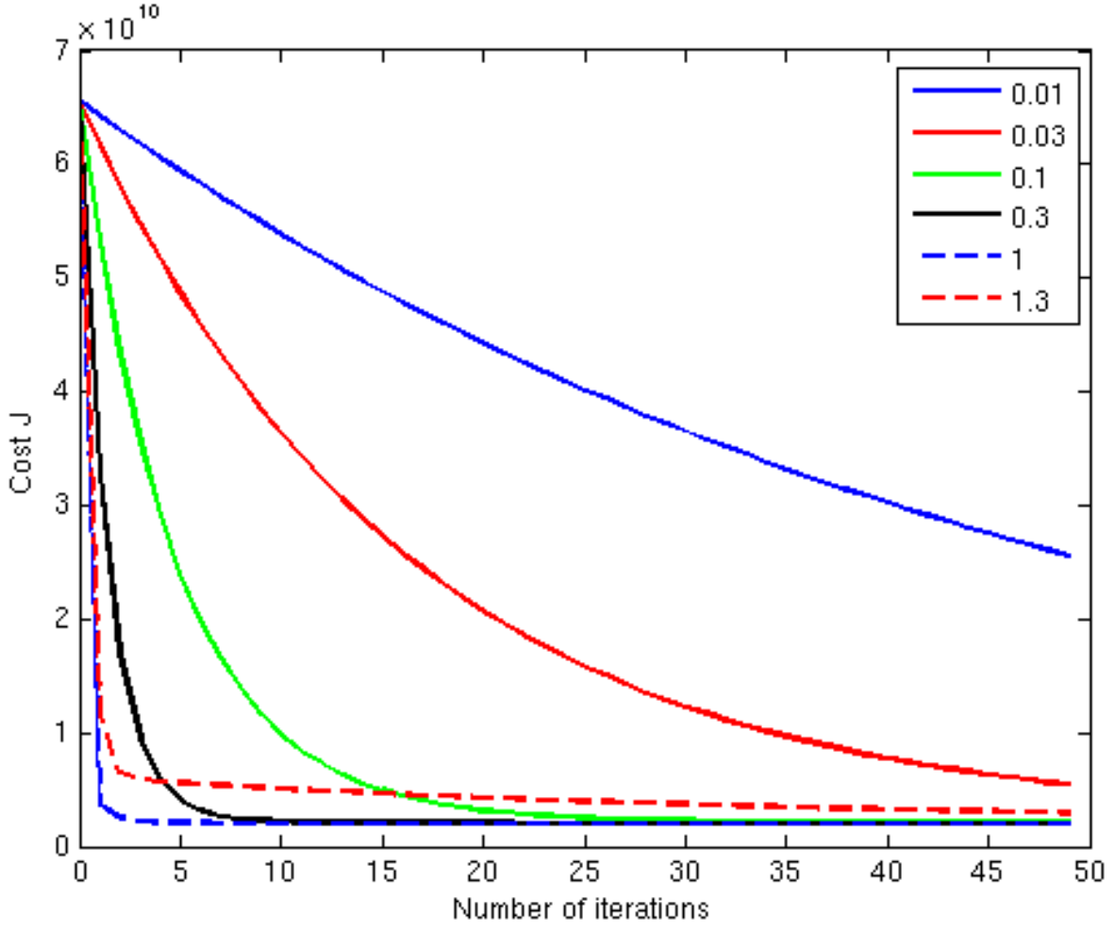
$$\theta_{n+1} = \theta_n - \eta F^{-1} \nabla \mathscr{L}(\theta)$$

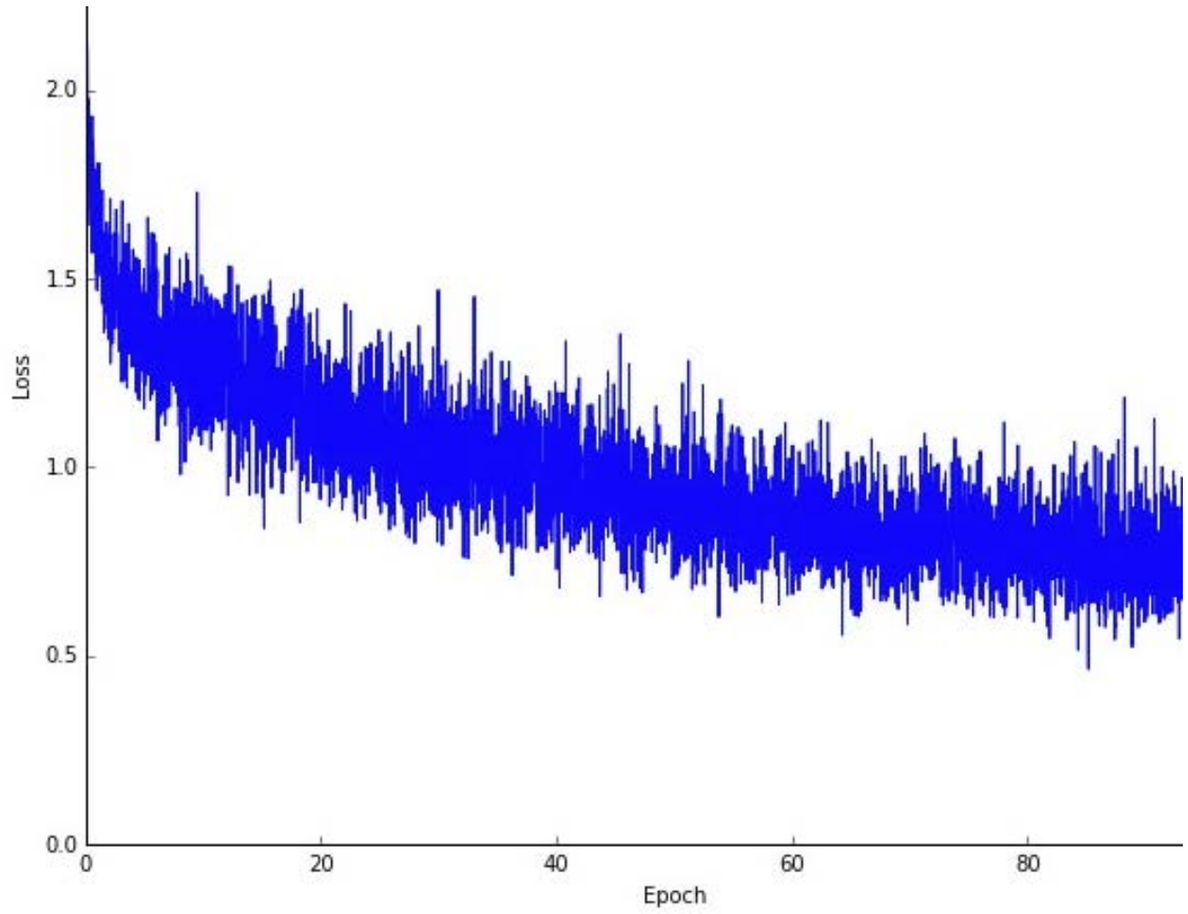
Cramer-Rao bound: The inverse of the Fisher information is a lower bound on the variance of any unbiased estimator of θ

Fisher Information Matrix == negative expected Hessian of log likelihood

$$\begin{aligned}
\mathrm{H}_{\mathrm{KL}[p(x|\theta) \,\|\, p(x|\theta')]} &= -\int p(x|\theta)\ \nabla^2_{\theta'} \log p(x|\theta')\big|_{\theta'=\theta}\, \mathrm{d}x \\
&= -\int p(x|\theta)\, \mathrm{H}_{\log p(x|\theta)}\, \mathrm{d}x \\
&= -\mathop{\mathbb{E}}_{p(x|\theta)} [\mathrm{H}_{\log p(x|\theta)}] \\
&= \mathbf{F}\,.
\end{aligned}$$

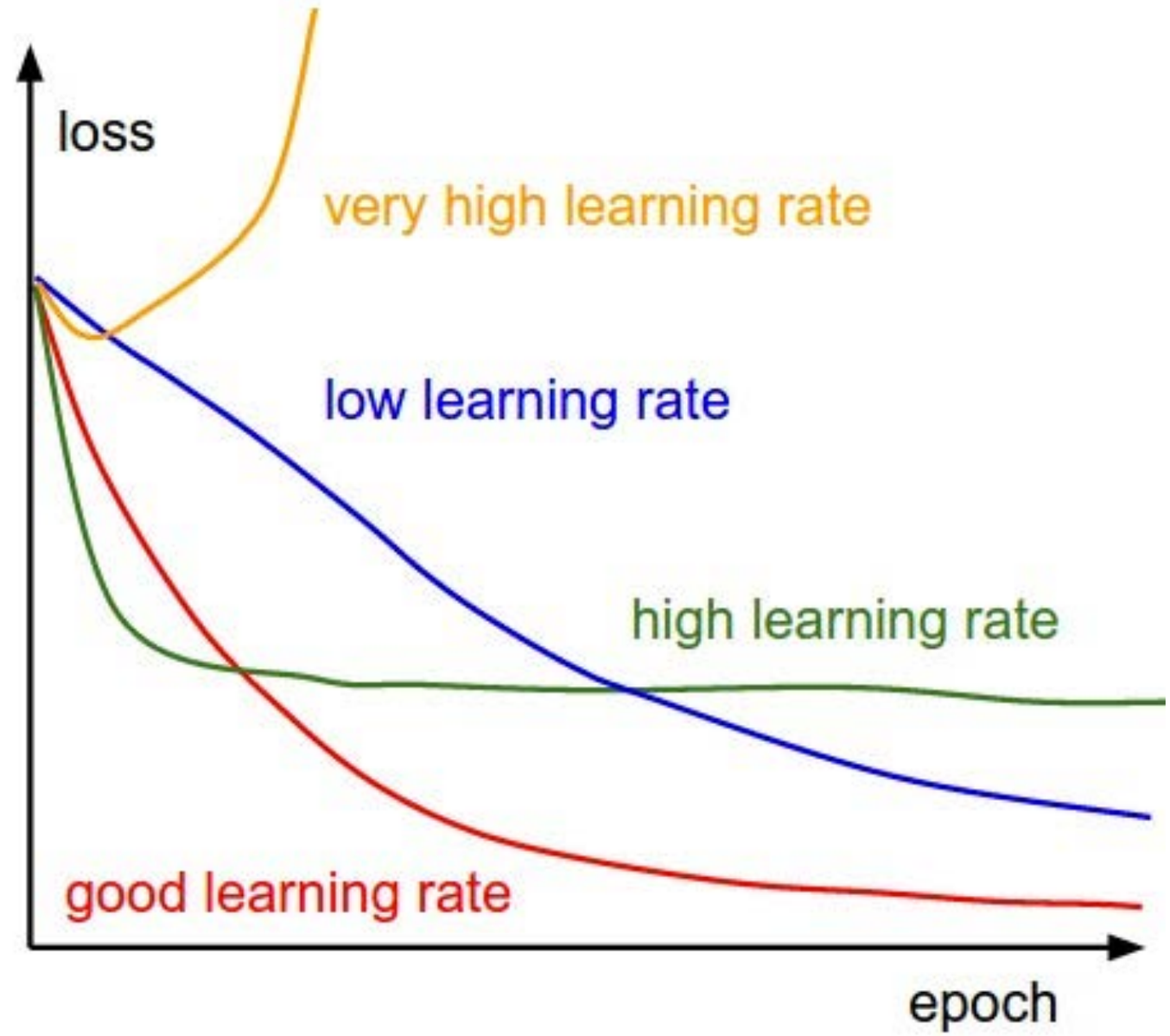
Gradient descent vs SGD





https://lossfunctions.tumblr.com

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.

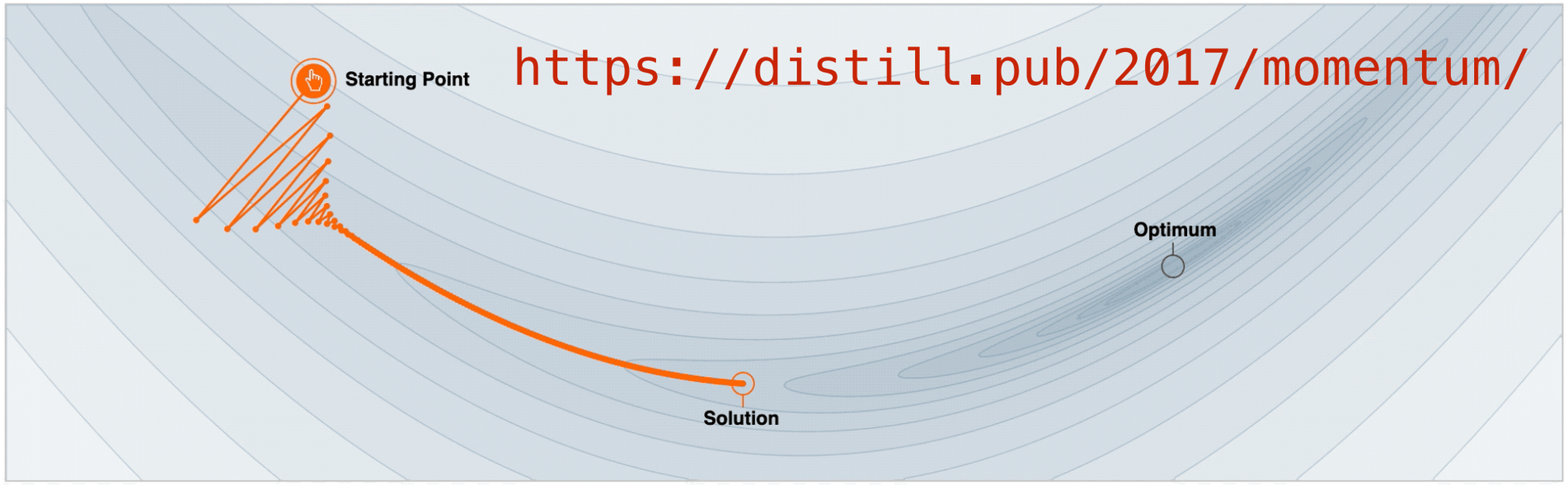
- The same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.

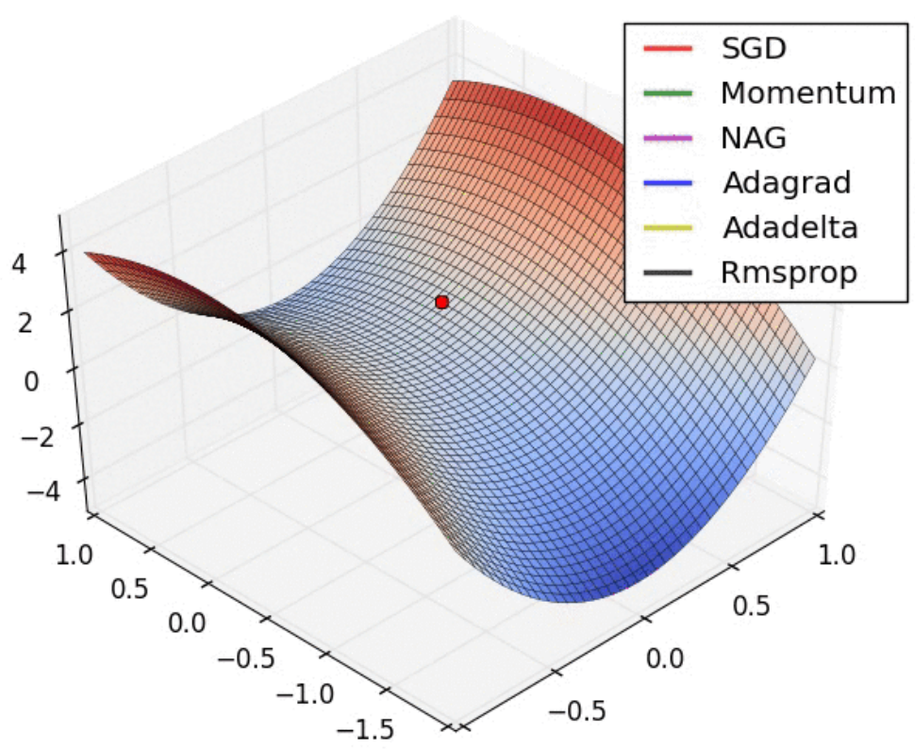
- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous suboptimal local minima. Dauphin et al. argue that the difficulty arises in fact not from local minima but from saddle points, i.e. points where one dimension 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.

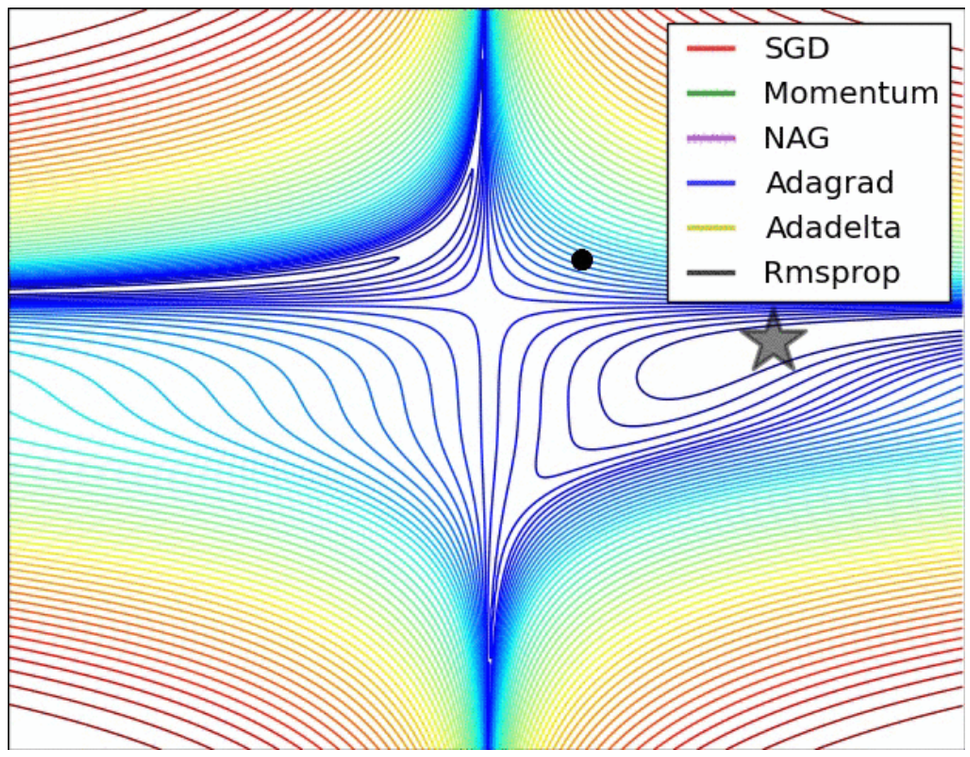
Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

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

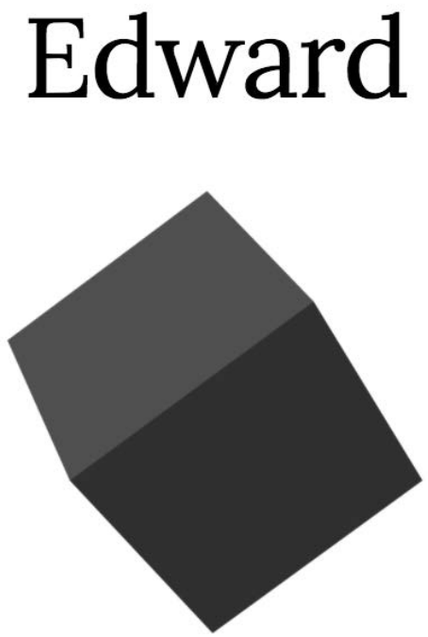
http://ruder.io/optimizing-gradient-descent/

*animation credit: Alec Redford

Probabilistic programming

http://mc-stan.org/

https://github.com/pymc-devs/pymc3



http://edwardlib.org/

https://github.com/uber/pyro