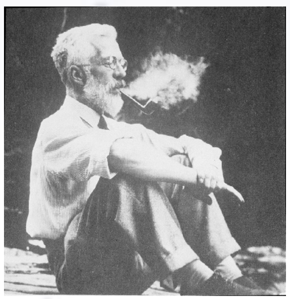
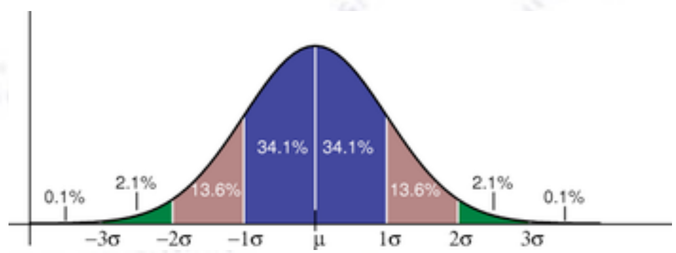
Applied ML

Stochastic Gradient Descent



Troels C. Petersen (NBI)



"Statistics is merely a quantisation of common sense - Machine Learning is a sharpening of it!"

(Normal) Gradient Descent

The choice of loss function, L, depends on the problem at hand, and in particular what you find important!

$$L(\theta) = \frac{1}{N}\sum_{i}^{N} L_i(\theta)$$

In order to find the optimal solution, one uses Gradient Descent **based on the whole dataset**:

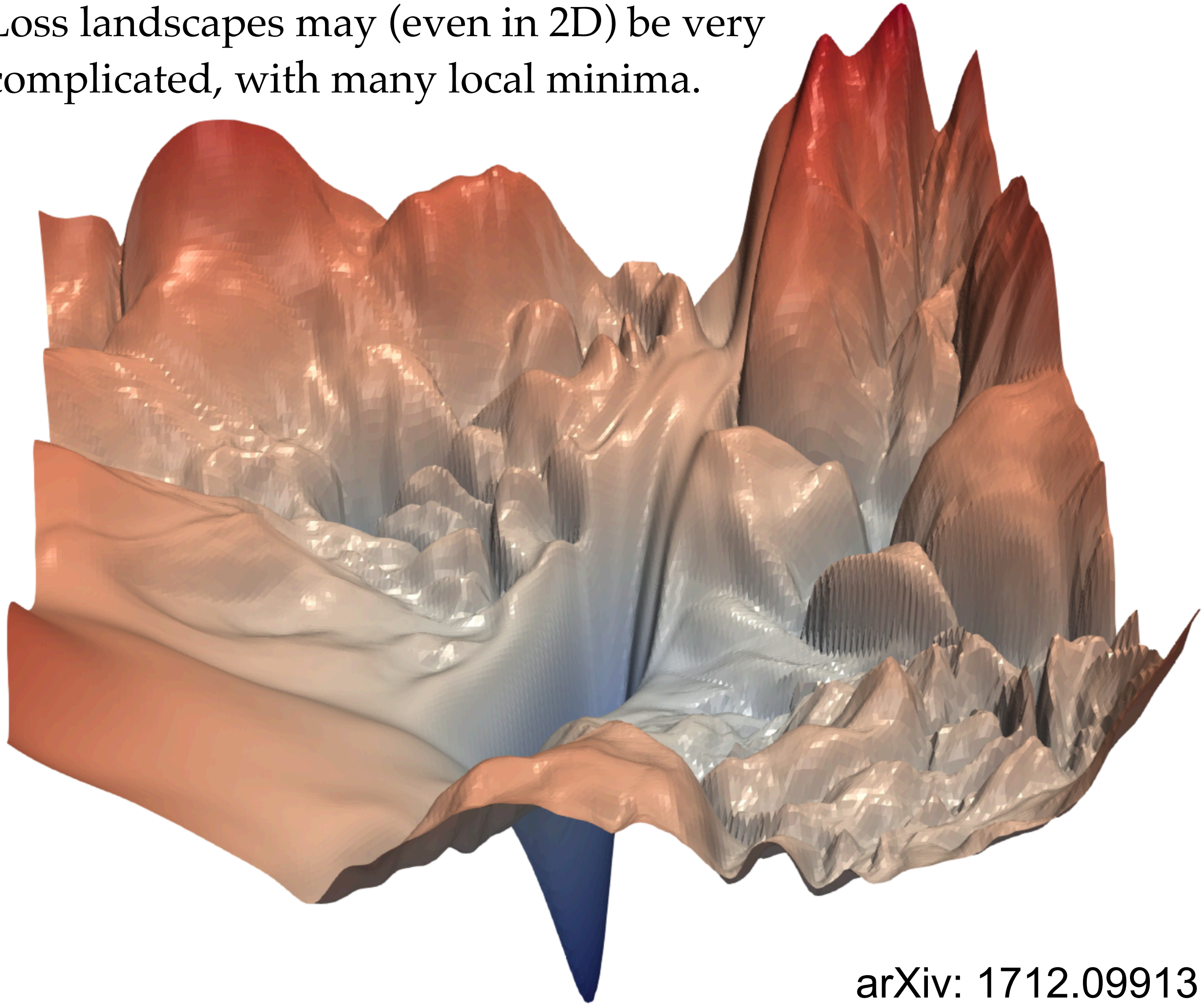
$$\theta_{j+1} = \theta_j - \eta\nabla L(\theta) = \theta_j - \frac{\eta}{N}\sum_{i}^{N} \nabla L_i(\theta)$$

This is the procedure used by e.g. Minuit and other minimisation routines.

Note the very important parameter: **Learning rate η**.

(Nasty) Loss Landscapes

Loss landscapes may (even in 2D) be very complicated, with many local minima.



arXiv: 1712.09913

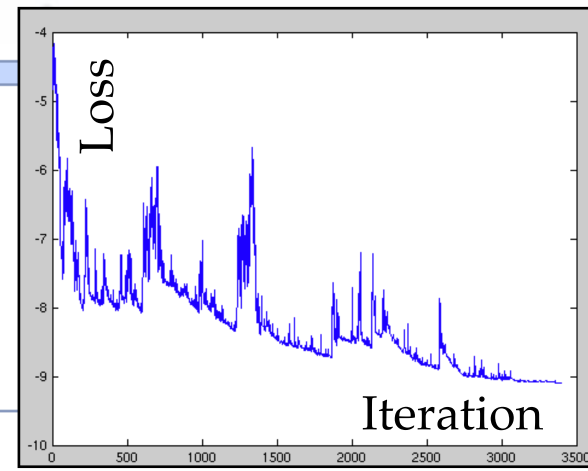
Stocastic Gradient Descent

In order to give the gradient descent some degree of "randomness" (stochastic), one evaluates the below function **for small batches** instead of the full dataset.

$$\theta_{j+1} = \theta_j - \eta \nabla L(\theta) = \theta_j - \frac{\eta}{N} \sum_{i}^{N} \nabla L_i(\theta)$$

The algorithm thus becomes:

- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta \nabla Q_i(w)$.



Not only does this vectorise well and gives smoother descents, but with decreasing learning rate, **it "almost surely" finds the global minimum** (Robbins-Siegmund theorem).

Example

Consider fitting a straight line $y_{hat} = \theta_1 + \theta_2$ x, given features (x_1, x_2, …, x_n) and estimated responses (y_1, y_2, …, y_n) using least squares. The Loss function is then:

$$L(\theta) = \sum_i^N L_i(\theta) = \sum_i^N (\hat{y}_i - y_i)^2 = \sum_i^N (\theta_1 + \theta_2 x_i - y_i)^2$$

To minimise this, we could consider stochastic gradient descent, where for each iteration, we only evaluate the gradient in a small batch (or single point):

$$\begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}_{j+1} = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}_j - \eta \begin{bmatrix} \frac{\partial}{\partial \theta_1}(\theta_1 + \theta_2 x_i^2 - y_i)^2 \\ \frac{\partial}{\partial \theta_2}(\theta_1 + \theta_2 x_i^2 - y_i)^2 \end{bmatrix}_j$$

The learning rate and batch size are the two Hyper Parameters, where it is the learning rate, that is the most important.

https://en.wikipedia.org/wiki/Stochastic_gradient_descent

Choosing Learning Rate

Too low learning rate: Convergence very (too) slow.
Too high learning rate: Random jumps and no convergence.
You want to increase it until it fails and then just below…

Ristet brød er let at lave
blot man vil erindre:
når det oser, skal det have
to minutter mindre.

Piet Hein

There's an art of knowing when.// Never try to guess.// Toast until it smokes and then// twenty seconds less.