[AIMLBD] MACHINE LEARNING, BIG DATA, ARTIFICIAL INTELLIGENCE per medicina e chirurgia high tech

L02: Optimization

Dott. Giorgio De Magistris

demagistris@diag.uniroma1.it

CORSO DI LAUREA IN MEDICINA E CHIRURGIA HIGH TECH

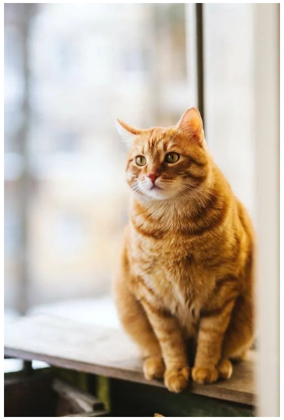
SAPIENZA
UNIVERSITÀ DI ROMA

I3S
FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE, INFORMATICA E STATISTICA

DIAG
DIPARTIMENTO DI INGEGNERIA INFORMATICA, AUTOMATICA E GESTIONALE

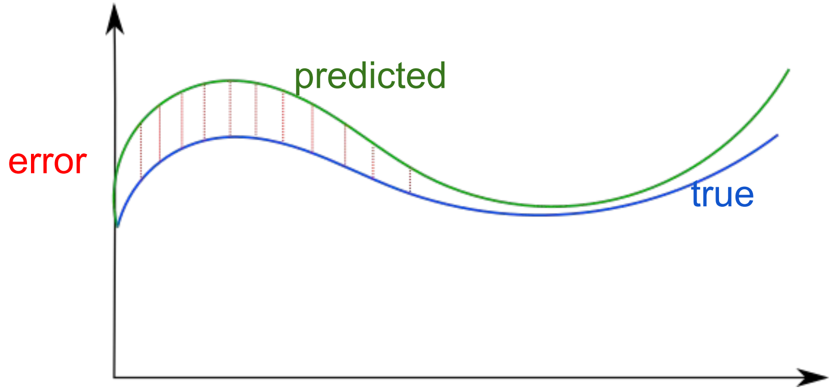
Optimization in Machine Learning

- In general a ML model is trained to minimize an error function with respect to the model parameters



Is this a mammal?

$(-1)^{ytrue} \log(ypred)$



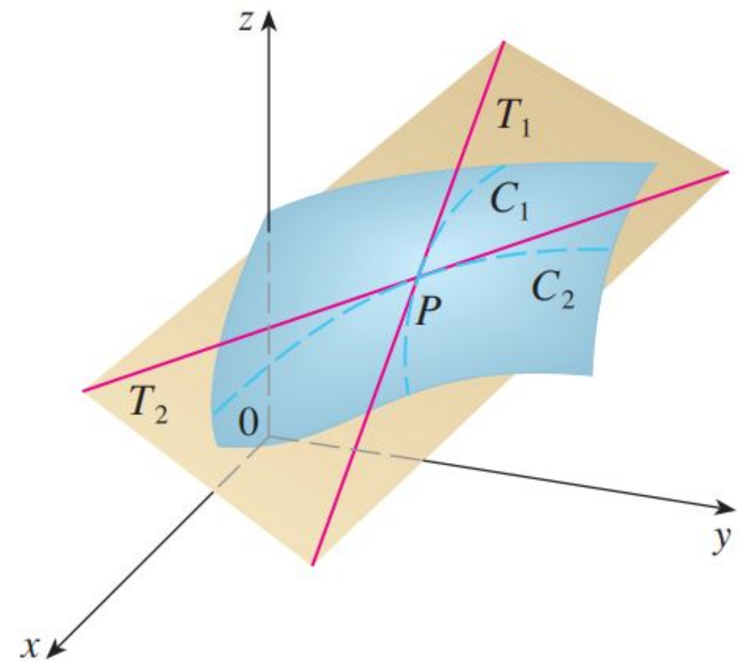
Partial Derivatives

- Suppose we have a function of two variables $z = f(x,y)$, (the blue surface in the figure)

- If I fix the value of one variable, say $y=y_o$ then I obtain a function of one variable $z = f(x,y_o)$ (the curve C_1 in the figure)

- The slope of the tangent of C_1 at a point $P(x_P,y_P)$ (T_1 in the figure) is the Partial Derivative of f with respect to x and it is indicated as:

$$\frac{\partial f}{\partial x}(x_P, y_P)$$



Minimization of a 2D function

P is a critical point if $$\bar{P}(\bar{x},\bar{y}) = \begin{cases} \dfrac{\partial f}{\partial x}(\bar{x},\bar{y}) = 0 \\ \dfrac{\partial f}{\partial y}(\bar{x},\bar{y}) = 0 \end{cases}$$

$$H_{\bar{x}_0} = \begin{pmatrix} \dfrac{\partial^2 f}{\partial x^2}(x_0, y_0) & \dfrac{\partial^2 f}{\partial x \partial y}(x_0, y_0) \\ \dfrac{\partial^2 f}{\partial y \partial x}(x_0, y_0) & \dfrac{\partial^2 f}{\partial y^2}(x_0, y_0) \end{pmatrix}$$

- Positive semidefinite → x_0 is min
- Negative semidefinite → x_0 is max
- Indefinite → x_0 is saddle

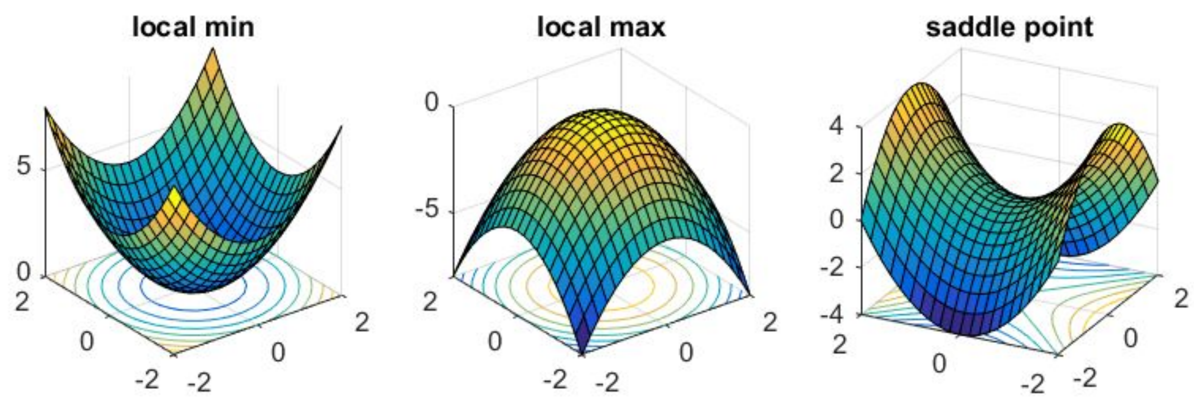


image credit: https://www.offconvex.org/2016/03/22/saddlepoints/

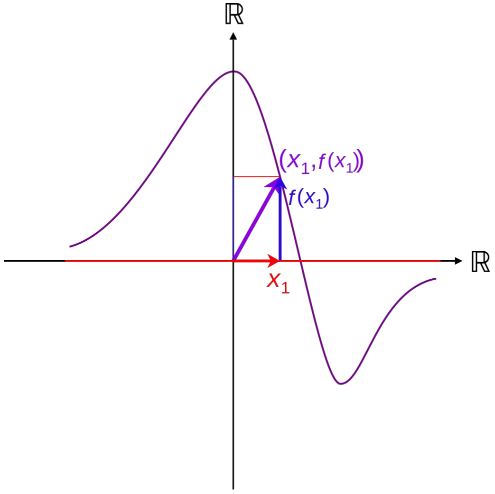
Minimization of an n-variable function

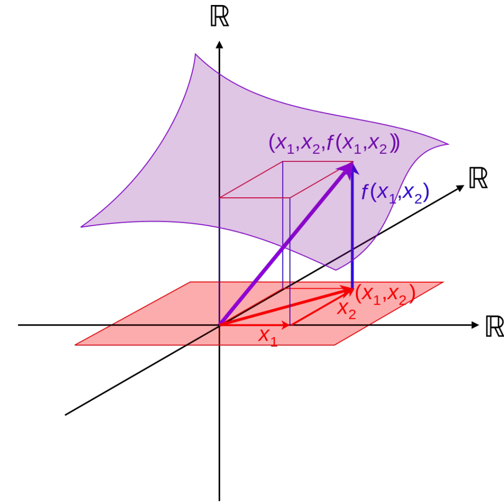
- Find critical points as before

$$\bar{\mathbf{x}} = \begin{pmatrix} \bar{x}_1 \\ \dots \\ \bar{x}_n \end{pmatrix} = \begin{cases} \dfrac{\partial f}{\partial \bar{x}_1} = 0 \\ \dots \\ \dfrac{\partial f}{\partial \bar{x}_n} = 0 \end{cases}$$

- Check whether max, min or saddle as before

- Closed form solution for the system of partial derivatives not always exists





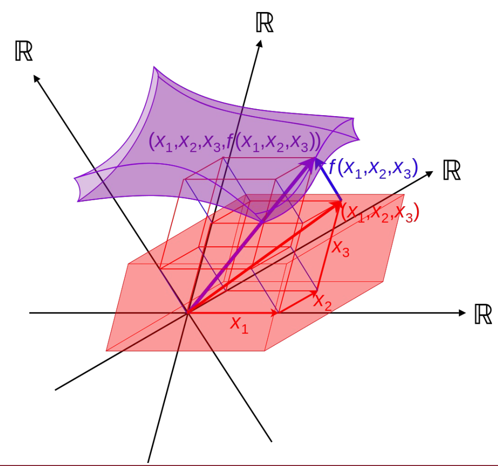


Image credit: [Functions of several variables]

Convex functions

- Given $\mathbb{X}$ a **convex subset** of $\mathbb{R}^N$ and the function
  - $f : \mathbb{X} \to \mathbb{R}$
- Then f is **convex** if

$$\forall \quad 0 \leq t \leq 1 \quad \text{and} \quad \forall \quad x_1, x_2 \in X \quad \text{then}$$

$$f(tx_1 + (1-t)x_2) \leq tf(x_1) + (1-t)f(x_2)$$

- Convex functions are easy to optimize, because they admit only a global minimum

convex subset of $\mathbb{R}^2$

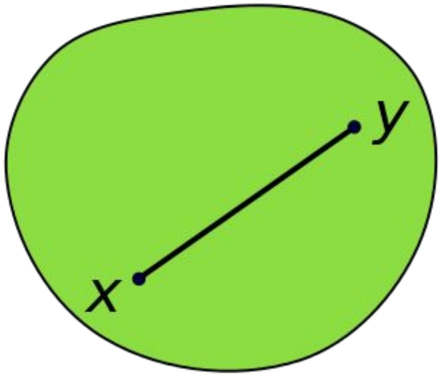
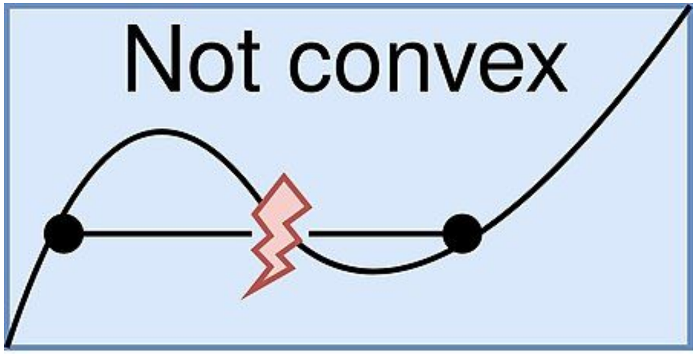


Image credit: https://en.wikipedia.org/wiki/File:Convex_polygon_illustration1.svg



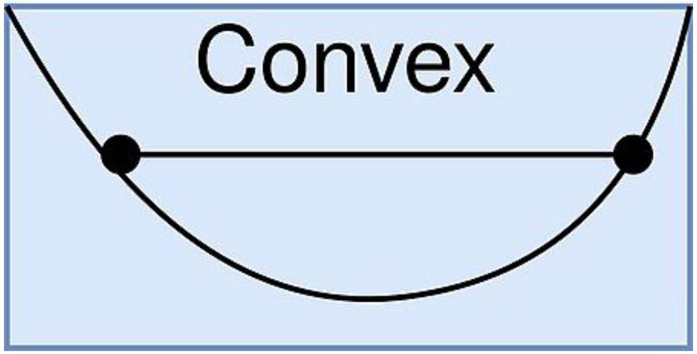


Image credit: https://en.wikipedia.org/wiki/File:Convex_vs._Not-convex.jpg

Gradient Vector

- The gradient vector at a point P is the vector of all partial derivatives at the point P
- It points in the direction in which the function increases the most
- The vector field that associates at each point in the domain the corresponding gradient vector is called the Gradient Vector Field

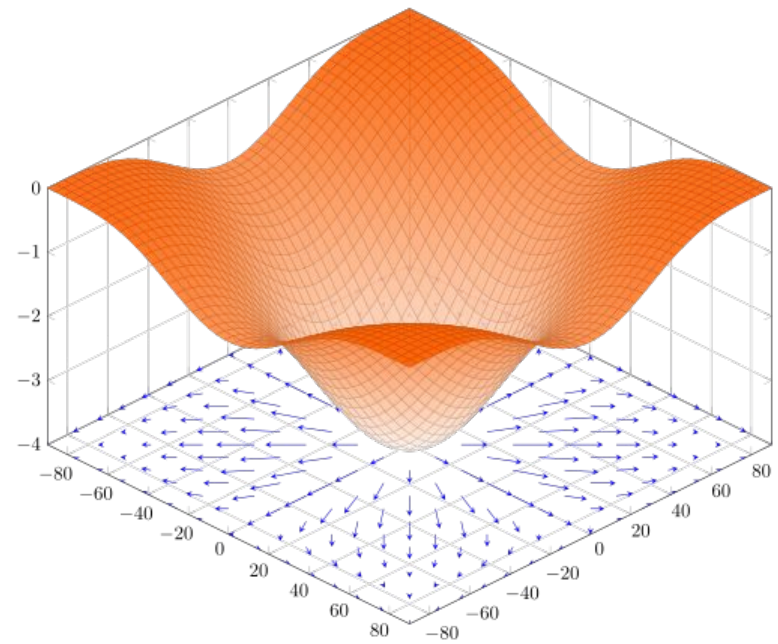


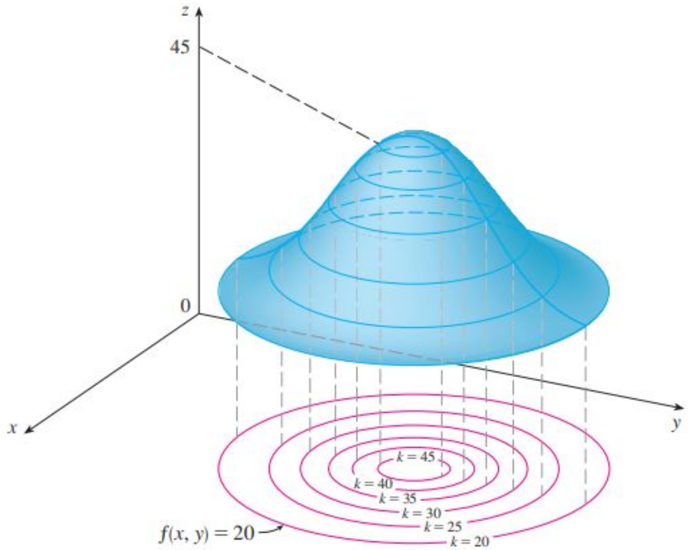
image credit: https://en.wikipedia.org/wiki/Gradient

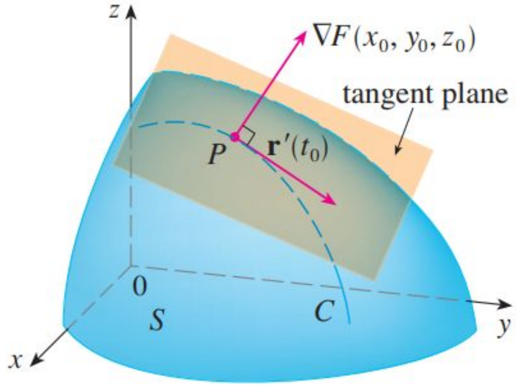
Level Curves

- The level curves of a function z = f(x,y) are the curves with equation f(x,y) = k where k is a constant in the range of f

- In other words the function f(x,y) is constant along the level curves

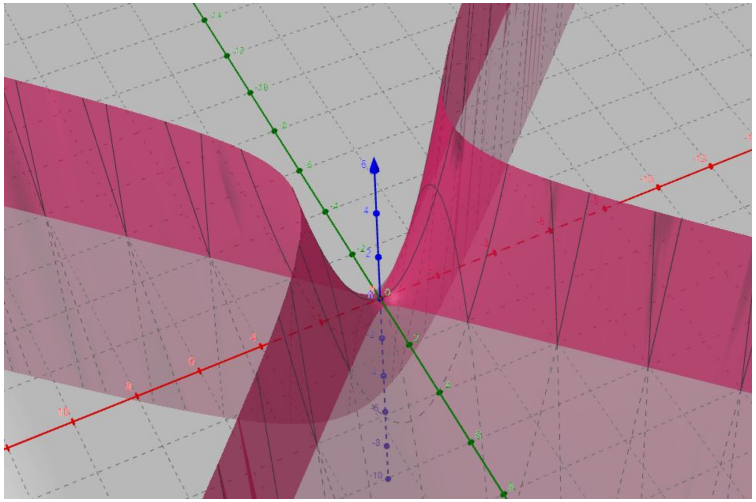
- For 3 or higher dimensional functions we talk about level surfaces (the blue surface in the right figure) or level hypersurfaces

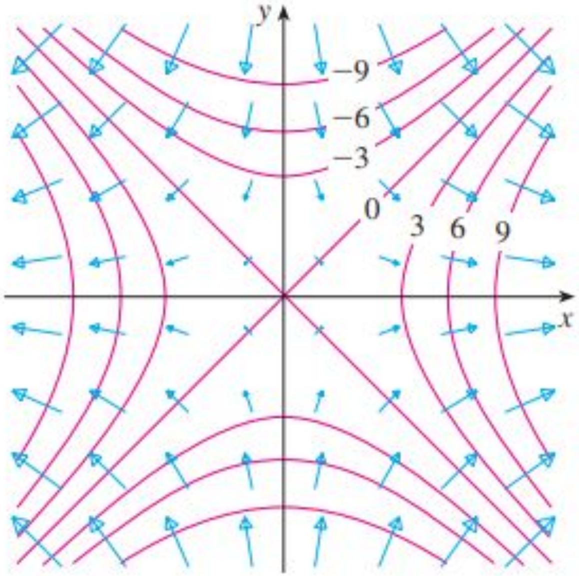




Gradient and Level Curves

- The gradient is always perpendicular to the level curves (or level hypersurfaces for higher dimensional functions)





Numerical Optimization

Notation: bold symbols (e.g. **x**) denote vectors

- In order to find the minimum $\mathbf{x}^*$ of $f(\mathbf{x})$:
  - Initialize $\mathbf{x}_0$ with a random point in the domain of f
  - Update **x** moving along a vector proportional to the inverse of the gradient vector (direction in which the function f decreases the most): $\mathbf{x}_{t+1} = \mathbf{x}_t - \delta_t \nabla \mathbf{f}(\mathbf{x}_t)$
  - Continue until convergence: $|f(\mathbf{x}_{t-1}) - f(\mathbf{x}_t)| < \varepsilon$

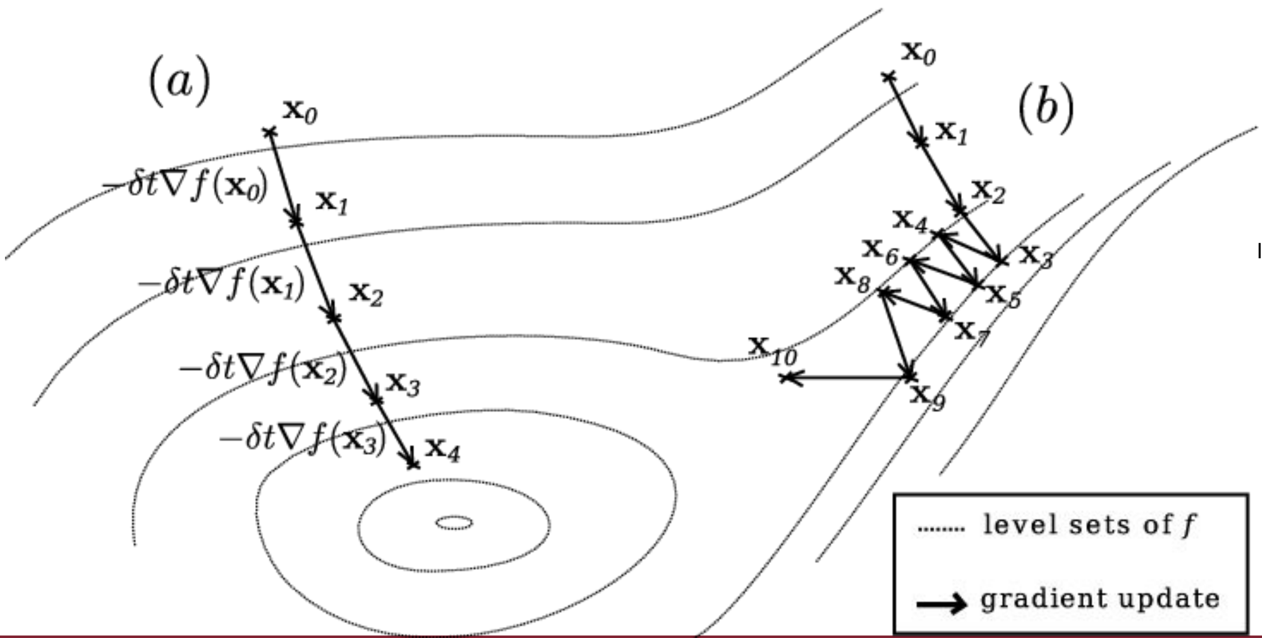


Image credit: [Gradient Descent]

Non convex optimization

- Often in machine learning the function to optimize with respect to the model parameters is not convex
- It could have many local minima, saddle points, flat regions
- Often the best we can do is local optimization, e.g. find local optima
- However often local optima works well on real data, while the global optima result in overfitting

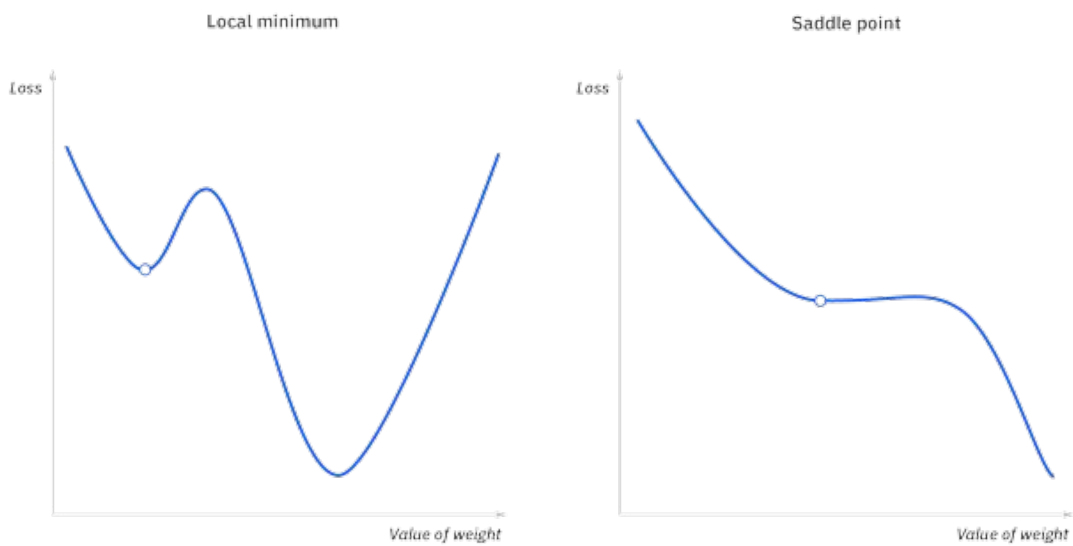


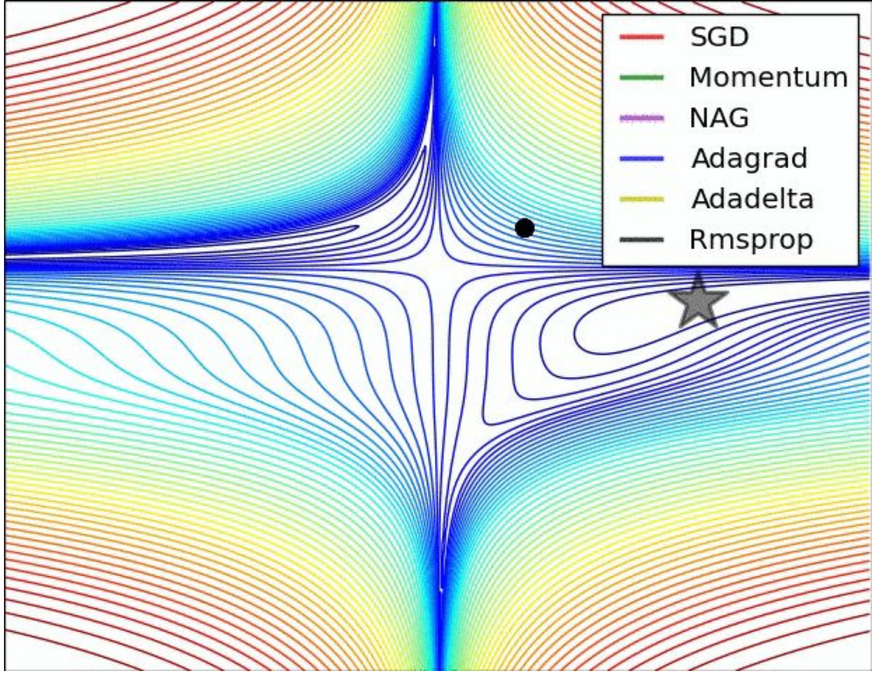
Image credit: [What is Gradient Descent?]

Optimization Strategy

- The optimization strategy is the rule that tells how to update the weights with respect to the gradients

- They are variations of the basic formula shown in the previous slide:
  - $\mathbf{x}_{t+1} = \mathbf{x}_t - \delta_t \nabla \mathbf{f}(\mathbf{x}_t)$

- Many different strategies exist and they have different convergence time and stability



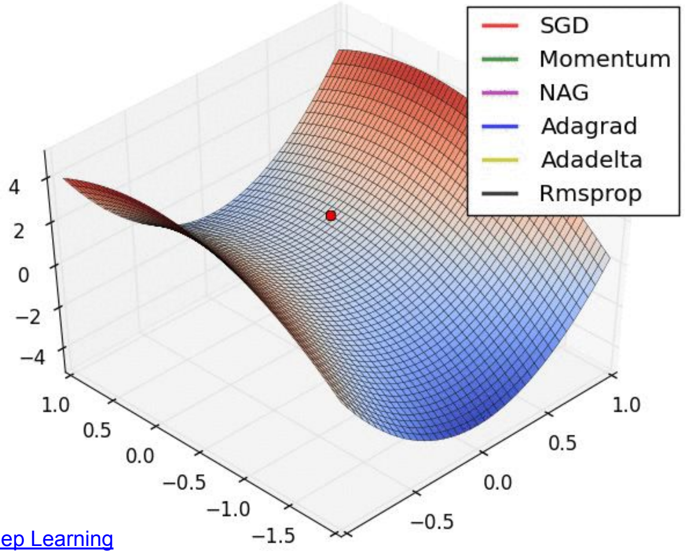
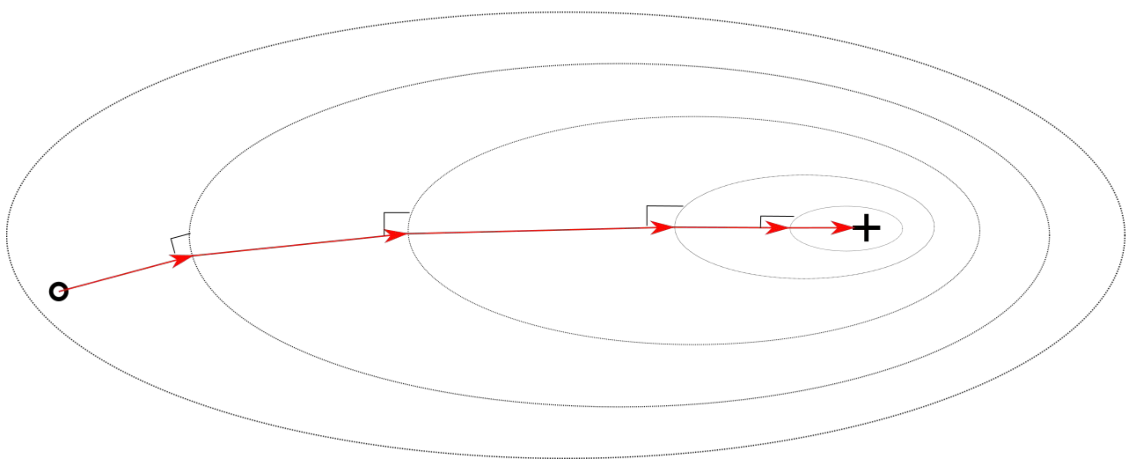


Image credit: [Optimizers in Deep Learning]

Gradient Descent

- In ML the function to optimize L(**D**,**θ**) is a function both of the trainingset **D** and the model parameters **θ**.
- In the gradient descent the parameters are updated according to the formula:
  - $\boldsymbol{\theta}_{new} = \boldsymbol{\theta}_{old} - \delta_t \nabla_{\boldsymbol{\theta}} L(\mathbf{D},\boldsymbol{\theta}_{old})$
- Note that for each update I have to compute the Loss function L for each training sample
- The resulting gradient is exact but the algorithm requires a lot of space and memory

$$\nabla_{\boldsymbol{\Theta}} L(\mathbf{D}, \boldsymbol{\Theta}) = \nabla_{\boldsymbol{\Theta}} \sum_{x \in \mathbf{D}} L(x, \boldsymbol{\Theta}) = \sum_{x \in \mathbf{D}} \nabla_{\boldsymbol{\Theta}} L(x, \boldsymbol{\Theta})$$



Stochastic Gradient Descent

- Similar to Gradient Descent but the gradient is not exact but rather it is estimated from a subset of elements sampled uniformly at random from the trainingset (mini-batch)
- The update formula is

$$\mathbf{\Theta}_{new} = \mathbf{\Theta}_{old} - \delta_t \nabla_{\mathbf{\Theta}} \frac{1}{n} \sum_{x \in \mathbf{B}} L(x, \mathbf{\Theta}_{old})$$

- Where B is the mini-batch and n is the size of the mini-batch
- The main advantage is that the optimization algorithm does not depend on the size of the dataset