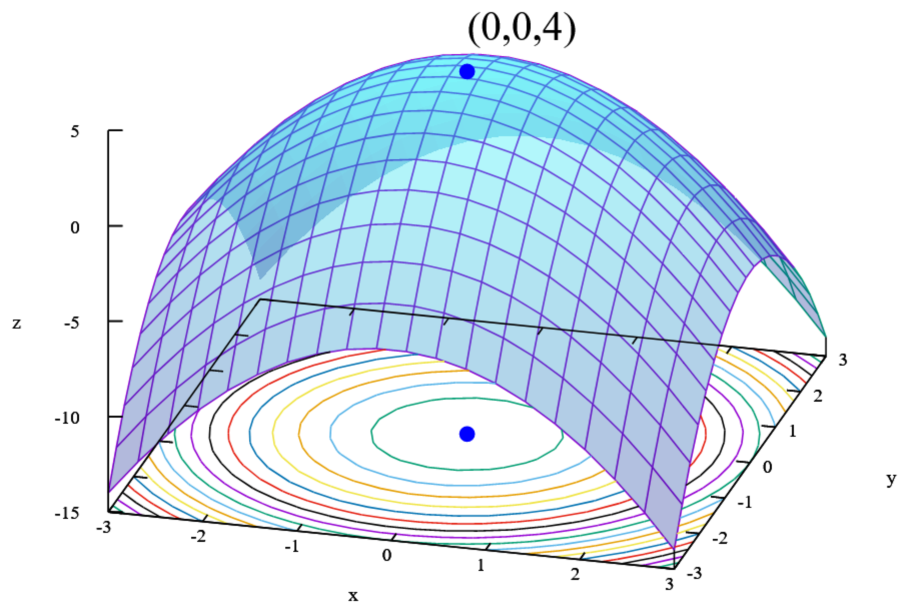
Machine learning optimisation – the short overview of most commonly used optimizers



Knowledge Sharing Session, March 2022

Introduction problem (prerequisites)

When training the model we usually try to find parameters of a function (a) to minimiza a loss function.

$$\sum_{i=1}^{n} L(y_i, \widehat{f_a}(x_{1i}, x_{2i}, \dots, x_{1k}))$$

Warsaw, Jan 2022

Introduction problem (prerequisites)

When training the model we usually try to find parameters of a function (a) to minimiza a loss function.

$$f_x(a) := \sum_{i=1}^{n} L(y_i, \widehat{f_a}(x_{1i}, x_{2i}, \dots, x_{1k}))$$

Warsaw, Jan 2022

Introduction problem (prerequisites)

Let's consider the following problem:

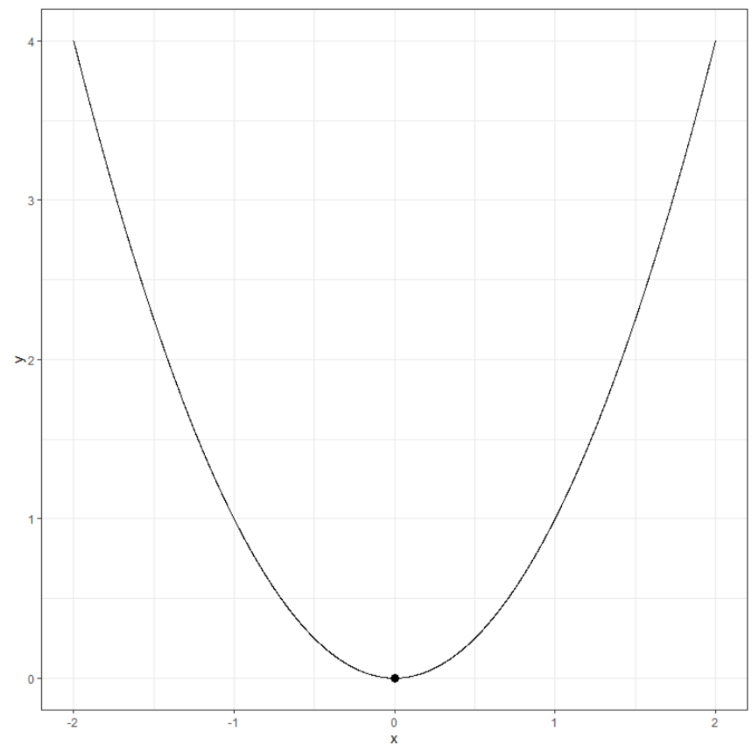
Given a function $D \rightarrow R$, we are supposed to find the argument x_{min} that minimizes the value $f(x)$.

$$x_{min} = arg \ \min_{x \in D} f(x)$$

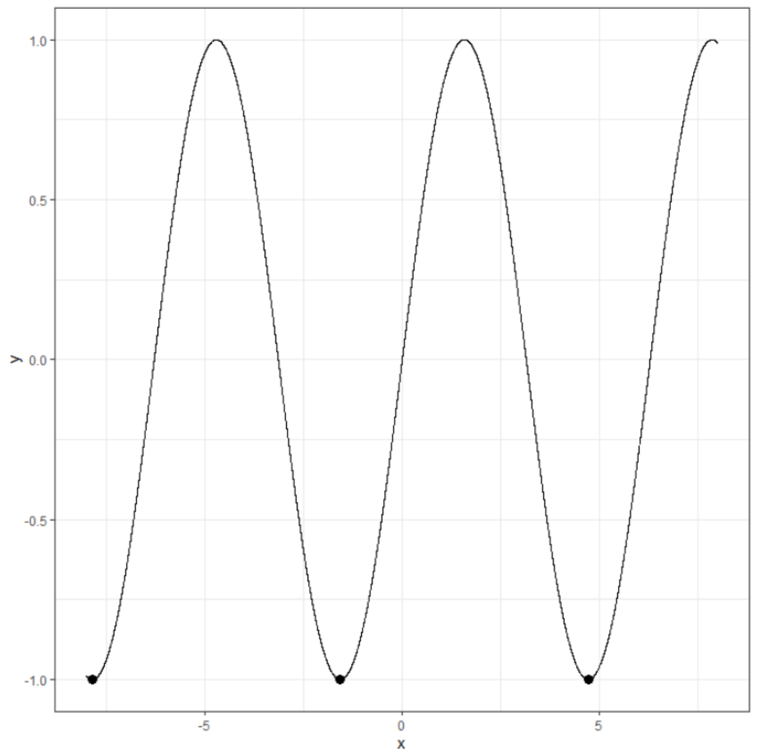
To find a maximum of function f, one can simply find the minimum of $-f$.

Warsaw, Jan 2022

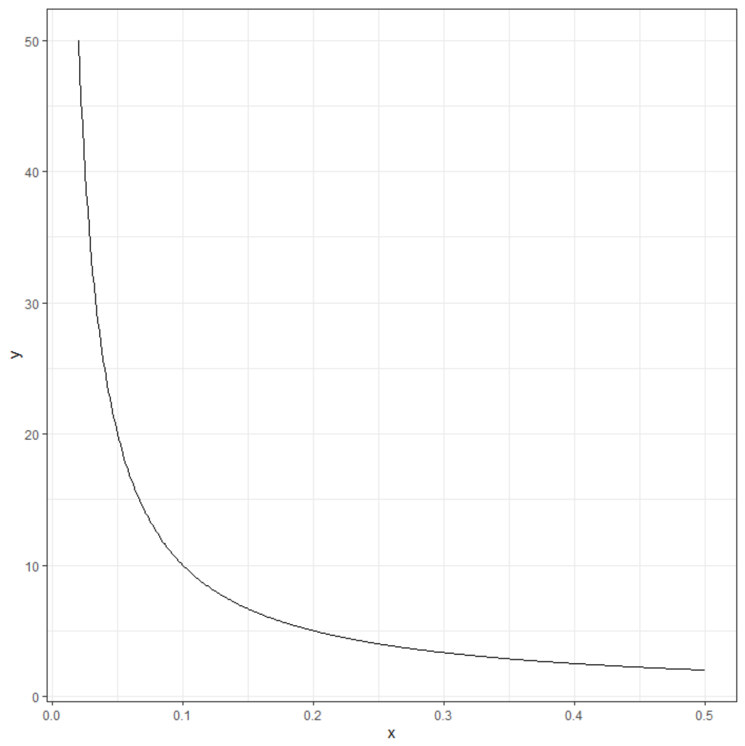
Introduction problem (prerequisites)



$$f: R \to R$$
$$f(x) = x^2$$

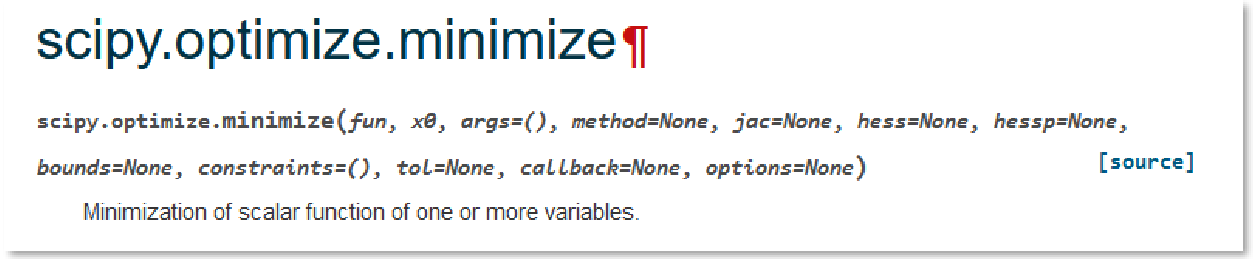


$$f: R \to R$$
$$f(x) = \sin(x)$$



$$f: R^+ \to R$$
$$f(x) = 1/x$$

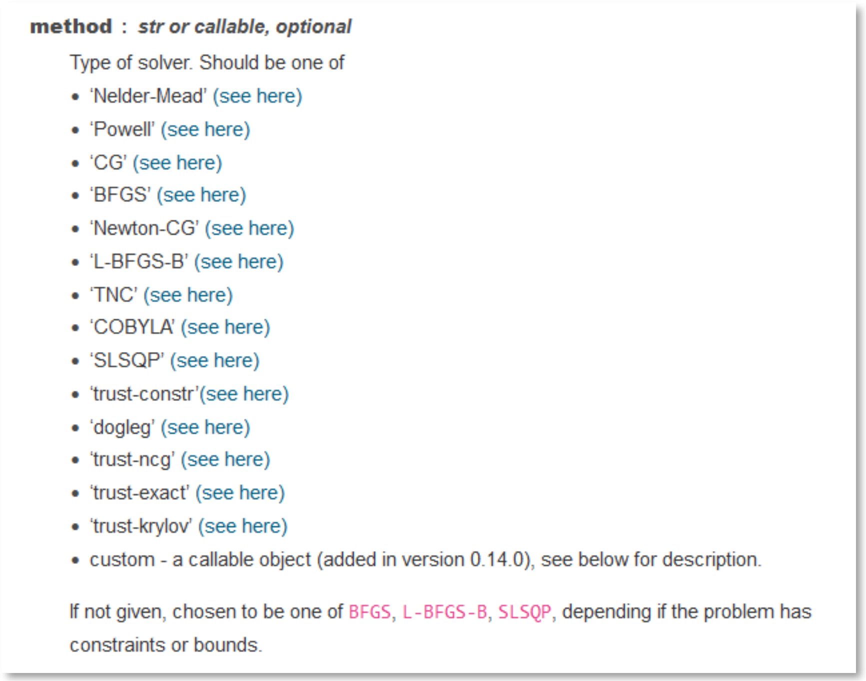
Introduction problem (prerequisites)



scipy.optimize.minimize¶

```
scipy.optimize.minimize(fun, x0, args=(), method=None, jac=None, hess=None, hessp=None,
bounds=None, constraints=(), tol=None, callback=None, options=None)
```
[source]

Minimization of scalar function of one or more variables.



method : *str or callable, optional*

Type of solver. Should be one of

- 'Nelder-Mead' (see here)
- 'Powell' (see here)
- 'CG' (see here)
- 'BFGS' (see here)
- 'Newton-CG' (see here)
- 'L-BFGS-B' (see here)
- 'TNC' (see here)
- 'COBYLA' (see here)
- 'SLSQP' (see here)
- 'trust-constr'(see here)
- 'dogleg' (see here)
- 'trust-ncg' (see here)
- 'trust-exact' (see here)
- 'trust-krylov' (see here)
- custom - a callable object (added in version 0.14.0), see below for description.

If not given, chosen to be one of `BFGS`, `L-BFGS-B`, `SLSQP`, depending if the problem has constraints or bounds.

Newton's method (requires twice-differentiability)

Nelder-Mead method (no differentiability assumptions)

L-BFGS (commonly default method, requires twice-differentiability)

https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html

Introduction problem (prerequisites)

Why these methods are currently not commonly used in machine learnig problems?

„Back in 2011 when that paper [L-BFGS] was published, deep learning honestly didn't work all that well on many real tasks. One of the hypotheses at the time (which has since been shown to be false) is the optimization problem that neural nets posed was simply too hard -- ***neural nets are non-convex, and we didn't have much good theory at the time to show that learning with them was possible. That's one of the reasons why people started exploring different optimization algorithms for neural nets****, which was a trend that continued roughly until the breakthrough results in 2012, which worked remarkably well despite only using SGD + momentum. Since then, more theory has been developed supporting this, and other tricks have been developed (BatchNorm, RMSProp/Adagrad/Adam/Adadelta) that make learning easier.”*

„It's because of ***memory issues*** *(e.g. LBFGS requires storing about 20-100 previous gradient evaluations) and more importantly* ***it does not work in stochastic setting*** *(e.g. minibatches which is very important since a full pass trough a dataset is very expensive and a lot of progress can be done with small minibatches). There have been many tryouts to make LBFGS work in stochastic setting, but none that I know to work well. In non stochastic case LBFGS is the best choice if memory does not come to be an issue.”*

https://www.reddit.com/r/MachineLearning/comments/4bys6n/lbfgs_and_neural_nets/

„*Fast large-scale optimization by unifying stochastic gradient and quasi-Newton methods*”

ML optimizers: basic idea

The general idea is to start with random point x_0 and (using only the function f) create the sequence of points

$$(x_0, x_1, x_2, x_3, \dots)$$

that converges to the local (global) minimum. Optimizers are nothing else then various algorithms of constructing the sequence above. The rate of convergence (how quickly does the sequence converge) is main our criteria of a good optimizer.

Optimizers require only the initial point and the function f to be optimised. Moreover, some of them have their own parameters that have to be set up. Computers „don't know" whether the function f is differentiable – it calculates the values of derivates assuming they exist.

For different functions the rates of convergence are different. Therefore, for a specific function f only one optimizer is the best one. But there is no one best optimizer in general for all functions.

For a specific funtion f, different optimizers have different rates of convergence (obvious). However, different optimizers can converge to different points, even if the starting point x_0 is the same!

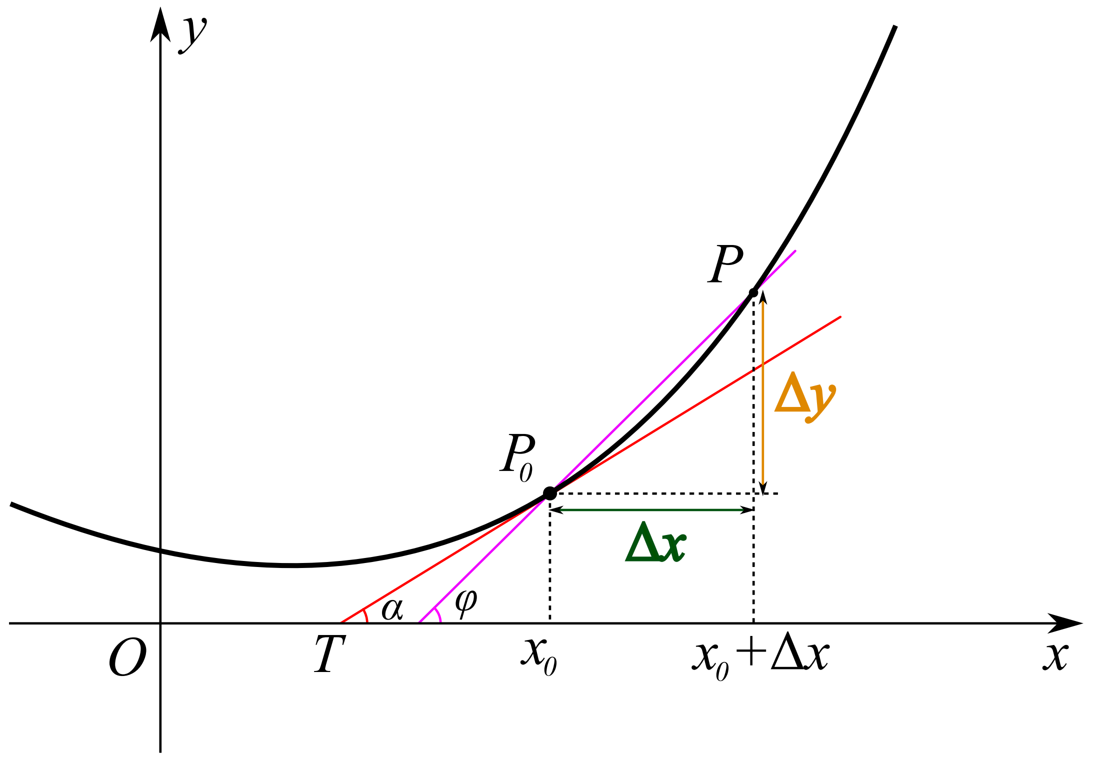
Available optimizers in Keras

Optimizers currently available in Keras Python library:

- SGD - (stochastic gradient descent) – the simplest, the most intuitive, and the first optimizer.
- RMSprop
- Adam - currently it the most commonly used optimizer in ML applications
- Adadelta
- Adagrad
- Adamax
- Nadam
- Ftrl

https://keras.io/api/optimizers/

Derivative interpretation



$$f'(x_0) \approx \frac{f(x_0+\Delta x) - f(x_0)}{\Delta x} = \frac{\Delta x}{\Delta y}$$

Δx - a very tiny change in x
Δy - a very tiny change in the output of f

f'(1) = 2.5: if we stand in the point x = 1, and make a very tiny step in the right direction, we will also have to make 2.5 steps up

f'(x0) > 0 – function is increasing at the point x0
f'(x0) < 0 – function is decreasing at the point x0