Stochastic Gradient Descent: Overview

Adam Loch, Szymon Wojciechowski

I. Introduction

This paper is presenting a general description of Stochastic Gradient Descent algorithm and results of work done to implement it. Authors prepared application and set of example images that can help to understand the algorithm way of working.

II. Stochastic Gradient Optimization

Stochastic Gradient Descent (SGD) - is an modification of gradient descent optimization algorithm, widely used for optimization of neural networks [4] or fitting regression models. The goal of algorithm is to minimize an multivariable function $J(\theta_0, \theta_1 ... \theta_n)$ for which denoted θ_n is an parameter of optimized model. Objective function must have a form:

$$J(\Theta) = \frac{1}{k} \sum_{i=1}^{k} Q_i(\Theta)$$

where Q_i can be associated with i-th observation in the dataset. The algorithm starts with randomly selected parameters θ. In each iteration (step) algorithm is updating parameters in such way:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

where η is a learning rate parameter of algorithm and $\nabla_\theta J(\theta)$ is gradient of the function. To understand that it can be said that algorithm is starting at some point on surface (generated by objective function) and in each iteration it is moving toward the direction where the descent is highest. It continues to do so until the satisfying minima is reached, or limit of iterations (calculation time) is reached.

The difference between stochastic gradient descent and generic (batch) gradient descent is that update in each iteration is calculated only on randomly selected subset of observations. It make a route to minima noisier than in full-calculations approach, but also it reduces amount of computations needed which makes it faster.

A. Gradient computation

It is essential to make gradient calculations for discussed algorithm. The general gradient formula can be defined as a vector of derivatives for each of the function variables x_n

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right]$$

Unfortunately, calculating derivatives in algorithm would require solving differential equation of whole formula, which in case of generic algorithm implementation is not trivial and sometimes impossible.

However it is possible to calculate numerical gradient using *Central-Difference Formulas* [2]. Assuming that $x \pm h$ is in neighbourhood of x, than

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Such assumption can simplify calculations but with a cost of truncation error: $O(h^2)$. It implies that all calculations are not precise, but as long as this value is small enough, algorithm will be sufficiently precise.

III. Goals

The goal is to prepare an application that will be able to present how algorithm is running optimization. It requires:

- Own implementation of SGD
- Own implementation of numeric gradient
- Visualization of steps performed by algorithm

To satisfy second requirements, it is needed that *Objective function must be a function of two variables*.

For programming authors selected Python language, which offers and efficient implementation of numerical computations [3]. It also provides a package for creating interactive plots [1] which will be used for algorithm visualization.

IV. Results

Created application is presenting to the user the animation, which updates on each step of algorithm. For presentation purposes, each step is intentionally delayed, to avoid reaching minima too early. The application can solve minimization problem on various functions. An example of output can be observed in Figure 1a and Figure 1b. Both Figures are representing same function - one as a surface and second one as contour lines.

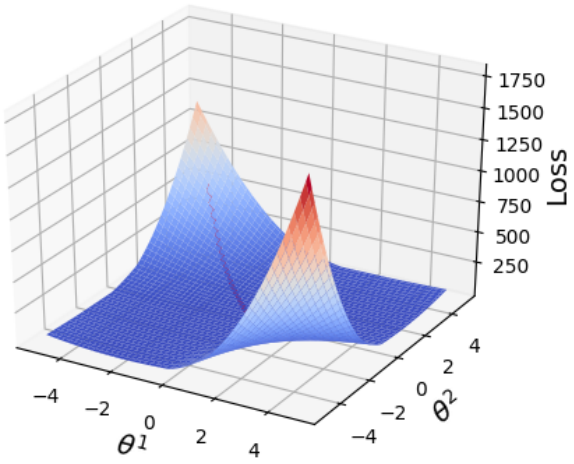
Also, there is red line which is drawing a route done to reach minima. Application can run on any device which has installed Python interpreter with `numpy` and `matplotlib` package.

Authors have used application to observe behaviour for two different goal functions.

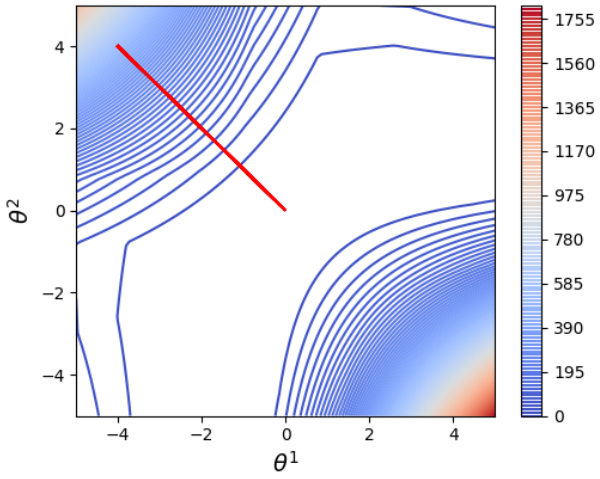
1) Mean Square Error: Mean Square Error (MSE) function is given by formula:

$$MSE = \frac{1}{n}\sum_{t=1}^{n} e_t^2$$

It is commonly used formula for fitting in a regression problem. Figure 1 presents plots generated by application. It can be observed that algorithm has moved from point `[4,4]` to `[0,0]` where it reached minima.



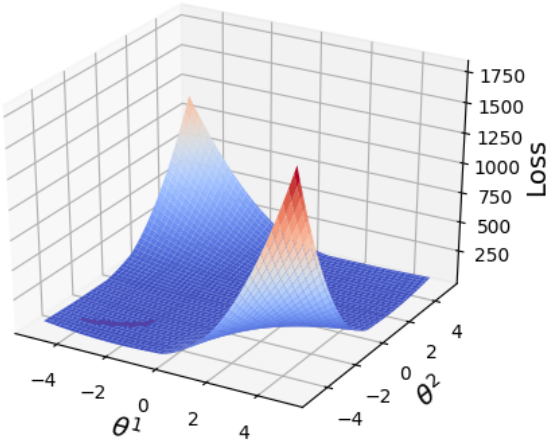
(a) 3D view on minimized function.



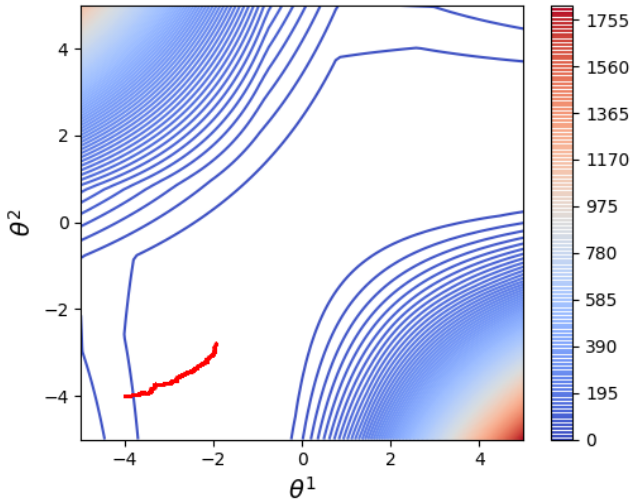
(b) Hypsometric view on minimized function.

Fig. 1: Plots of MSE - Example 1.

Another example, presented in Figure 2 shows same function, but this time algorithm start at the point `[-4, -4]` and reaches minima at `[-2, -2.2]`.



(a) 3D view on minimized function.



(b) Hypsometric view on minimized function.

Fig. 2: Plots of MSE - Example 2.

Comparing both paths it can be observed that in first example algorithm was heading straight forward to minima, as it was falling from heap. Second example is starting in function plain and steps are small, but globally can be considered chaotic.

This examples show that algorithm is moving as fast as the function descent. Also, when descent is small, algorithm starts to make small moves in various directions. The reason for this is that SGD is using subset of all observations, which comes with cost of loosing the precision.

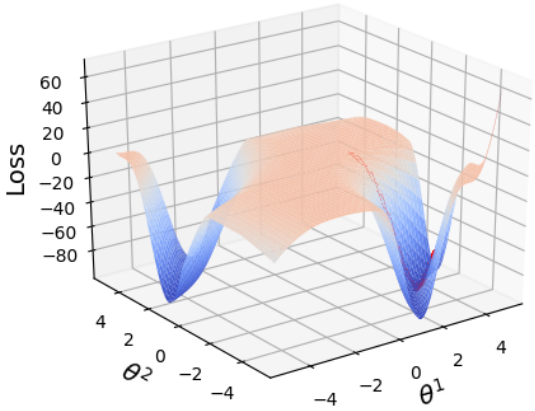
2) Custom Function: To also observe non standard function, authors have defined a Custom Error Function (CEF) and it is given by formula:

$$CEF = \frac{1}{n}\sum_{t=1}^{n} sin^3(e_t)$$

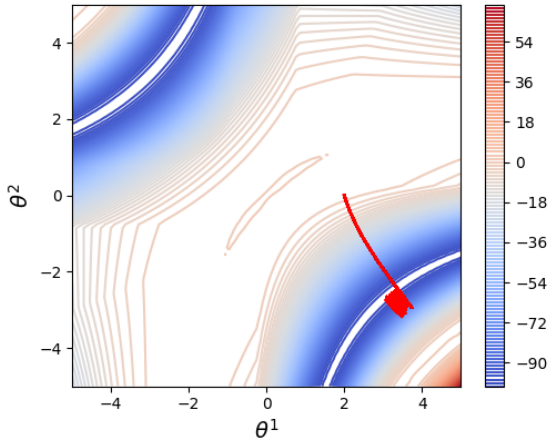
This function does not have any real life appliance, but in case of this work, it can be used as an interesting abstract model.

In Figure 3 it can be observed the plot of such function. Very characteristic for periodic function sin used in formula are many minimas (only two visible in this example) repeating in whole domain.

In this example algorithm start in point `[2, 0]` from which it moves to `[3, -2.2]` to one of minimas. When local minima is reached algorithm starts to float around local minima drifting forth and back to it.



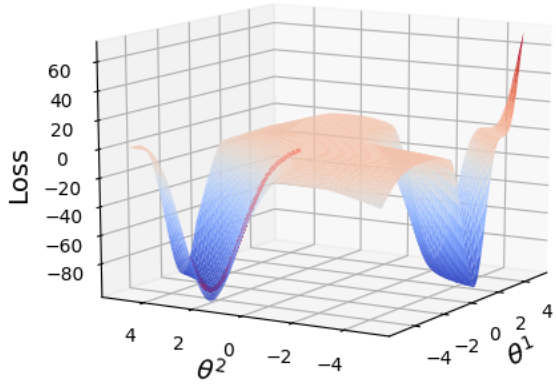
(a) 3D view on minimized function.



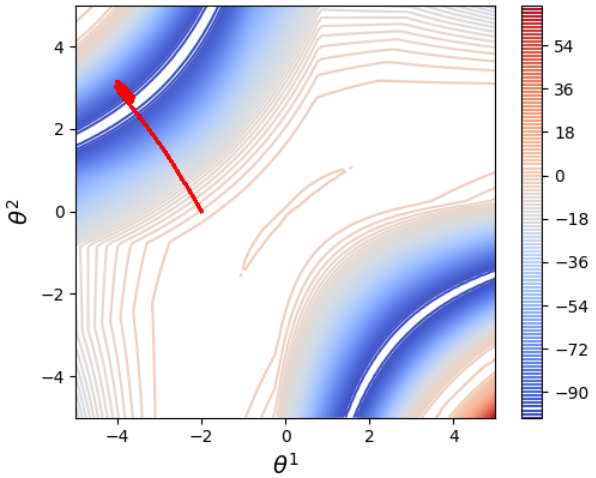
(b) Hypsometric view on minimized function.

Fig. 3: Plots of CEF - Example 1.

The second example shown in Figure 4 show same function, but starting point is set to `[-2, 0]`. Algorithm has reached the other minima in `[-3.8, 3]`. The path thou, is the same - algorithm rapidly reaches the minima and starts to drift around it in final steps.



(a) 3D view on minimized function.



(b) Hypsometric view on minimized function.

Fig. 4: Plots of CEF - Example 2.

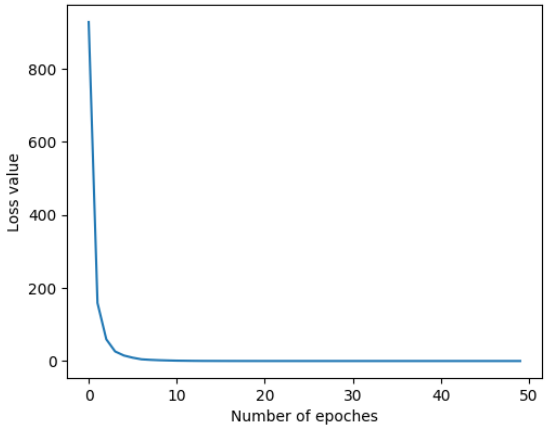
This example is showing an impact of starting point to the algorithm - by selecting two different points, both runs have ended up with two different solutions for this optimization. Of course, both results are correct and both should not be surprising. As it was mention before - goal function is periodic, so minimas will be repeating all over the domain.

This function is also unique, because decent is not linear as it was in MSE example. It causes slowing algorithm movement on lower descent, but with proper number of epoches minima is reached any way.

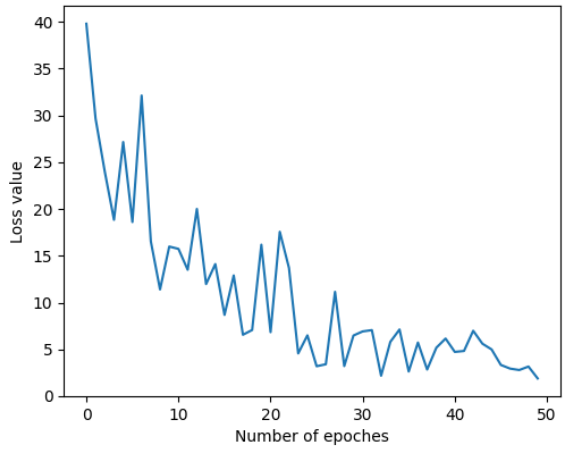
A. Loss Value over epoches

Another interesting aspect which can be observed using application is behaviour of Loss function over epoches. Plots in Figure 5 and Figure 6 are presenting results of MSE and CEF from previous examples accordingly.

In Figure 5a the descent starts at 800 and it is very fast and very stable - each step is minimizing objective function. However, in Figure 5b the descent is chaotic, but the values are starting from lower level (40) than in first example.



(a) Example 1

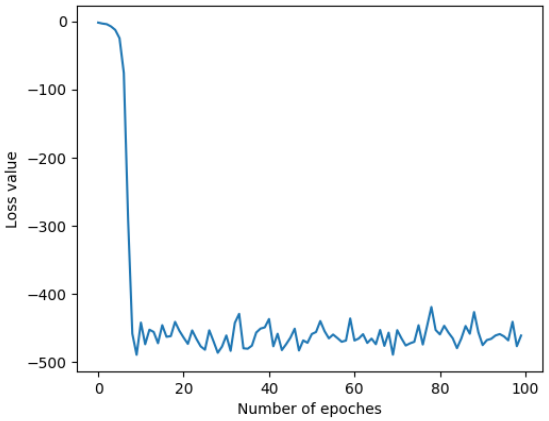


(b) Example 2

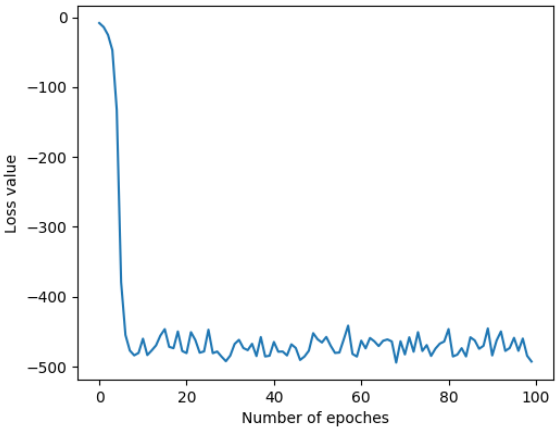
Fig. 5: Plot of loss value of MSE over epoches

In Figure 6a and 6b the shape of descent seems to be the same while, as it was prsented before, algorithm has reached two differen minimas. The descent is not linear, it starts slowly, and than accelerates very fast. At 10-th epoch is reaches certain level and it drifts around it.

In both cases, interpretation of plots is similar and it is reflecting tendencies of algorithm route. Such plots can be used to better analyse algorithm not only in cases



(a) Example 1



(b) Example 2

Fig. 6: Plot of CEF loss value over epoches

covered by this work, but also more generic ones where number of variables is high and visualizing the path is impossible.

V. Conclusion

The Stochastic Gradient Algorithm gives a method for optimizing parameters of given function very fast. By taking advantage of probability, it allows to reduce a computational resources, especially on large datasets. In addition, animations rendered by application, can give an intuitive example to understand idea hidden by gradient descent methods.

References

[1] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[2] John H. Mathews and Kurtis D. Fink. *Numerical Methods Using MATLAB*. Simon & Schuster, Inc., New York, NY, USA, 3rd edition, 1998.

[3] Travis Oliphant. *Guide to NumPy*. Trelgol Publishing, 2006.

[4] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.