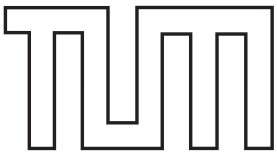


Technische Universität München
Chair of Theoretical Information Technology
Prof. Holger Boche

Topics in Optimization for Data-Driven Applications
Accelerated Nesterov

5 February 2020

Author: Mariem Khlifi
Matriculation Number: 03709738

Contents

1 Introduction

Optimization is a problematic at the core of different fields, be it in computer science, communication engineering or optics. It usually takes the following form:

$$minf(x)$$

The need to find the parameters that will ensure an optimal behaviour has always been existing because we do not dispose of unlimited computational resources nor execution time. For this reason, many methods were engineered to solve this problematic while overcoming all the mentioned problematics.

There are two major groups of methods in optimization that can be classified into first order and second order methods. Some famous first order methods have shown weaknesses when it comes to dealing with saddle points and local extrema. The simplicity of the first order methods comes with a price as it's practically impossible once you land in one of these saddle points to escape from them. This is where second order methods show their strength, because the use of the hessian helps in adding another consideration to detect whether an extrema was actually encountered or not.

Even though it seems that with second order methods, we should have our issues solved, it's unfortunately not the case. In fact, current data-driven applications use a lot of data and it's impossible with the tools at our disposal to rely on second order methods. In fact, the computation of the hessian needed for these second order methods can be very expensive as storing the matrix alone would need around 3725GB of memory for 1 million parameters[1]. Also, the second order methods are not compatible with mini-batches, a very prominent notion in machine learning.

For these reasons, it is safe to say that our focus should remain on first order methods. Fortunately, there are variations of the vanilla gradient that overcome the issue of performance which come with the first order methods. The solution to this was first introduced through momentum updates in the form of a classical momentum update or its variation, a better performing method under the name of Nesterov Accelerated Gradient.

The very idea of momentum was actually born from derivations from a second-order ordinary differential equation (ODE) but the modifications brought with the Nesterov Accelrated Gradient are yet to be explained using an ODE.

Through this present review, we will first introduce the idea of momentum and the Nesterov Accelerated Gradient. We will also get to see the advantages that the momentum update brings to many applications. We will then introduce some of the insights that will help us understand the principles behind the behaviour and the performance of the Nesterov Accelerated Gradient through the usage of a second-order ordinary differential equation[2].

Even though optimization methods that work for both non-convex and convex functions exist, we will put our focus on the optimization of convex functions going until exploring the specificities of strongly convex functions and how to ensure that the desired properties will hold.

2 Momentum Update

The basic Gradient Descent can be replaced with momentum updates for a better performance. The idea of Gradient Descent is still there but enhanced to overcome the convergence issues. It seems that the improvements are not only in escaping the wrong points but improving accuracy and time needed to reach the optimizer.

2.1 The Idea behind the Momentum Update

What characterizes the Momentum Update is the memory term that is added. The memory term is important for specific applications like Recurrent Neural Networks where past events should be remembered for as long as possible[3]. We can see the update rule below:

$$\begin{cases} x_{k+1} = x_k + \nu * v_k \\ v_k = \mu * v_{k-1} - \nabla f(x_k) \end{cases}$$

We can indeed see that this is derived from the classical Gradient Descent when we replace μ with 0. We will then end up with:

$$x_{k+1} = x_k - \nu * \nabla f(x)$$

The figure 1 visually explains what happens when Momentum Update is used instead of a vanilla Gradient. We clearly notice that the step resulting from the momentum advances faster. We can also see from the momentum update set of equations that terms with the indices k, $k-1$ and $k+1$ all appear at once which supports the reason behind calling the momentum term **the memory term**.

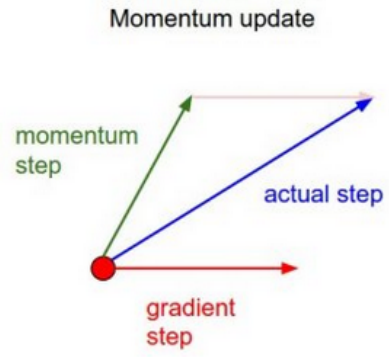


Figure 1: Explanation of the Momentum Update[1]

2.2 Nesterov Accelerated Gradient

Nesterov Accelerated Gradient is a variation of the Momentum Update. The update rule has a different principle of how to execute the look-ahead. The update rules are described below:

$$\begin{cases} x_{k+1} = x_k + v_k \\ v_k = \mu * v_{k-1} - \nu * \nabla f(x_k + \mu * v_{k-1}) \end{cases}$$

The figure 2 visually explains the principle behind the Nesterov Accelerated Gradient when it comes to choosing the look-ahead point at which the gradient will be evaluated.

The difference between the Momentum Update and Nesterov is that the former changes the direction of descent by looking farther from the current evaluated gradient using the momentum step whereas Nesterov evaluates the gradient at a look-ahead point to be sure that there is no strong variation from the direction of descent when the momentum step is later added.

Both provide similar performances but Nesterov performs better in certain situations and this is thanks to the gradient direction correction.

We will show in 4 that the factor $\sqrt{s}$ is responsible for making Nesterov faster than the normal Gradient where a factor of s is introduced. Considering that s is small, $\sqrt{s}$ is even smaller and allows a faster navigation that reaches the optimum under a smaller number of iterations as depicted in the figure 3.

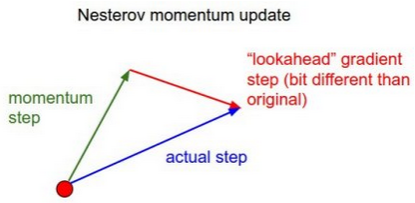
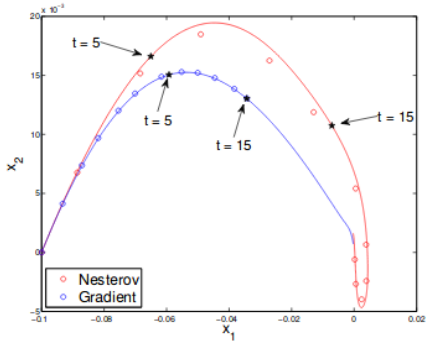


Figure 2: Explanation of the Netserov Update[1]



(b) Race between Nesterov's and gradient.

Figure 3: Comparison between the normal gradient and Nesterov[1]

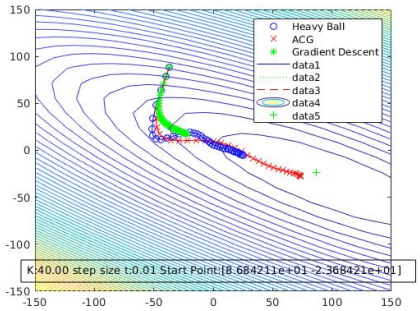


Figure 4: Comparison of the convergence of different methods[4]

3 Applications of Nesterov Accelerated Gradient

In this part, we are going to discuss some of the simulations that were run to show what characterizes the Nesterov Accelerated Gradient in comparison with the classical Gradient Descent.

Figure 4 shows the performance of the three commonly used methods: Gradient Descent(green), Momentum Update(blue) and Nesterov Accelerated Gradient(red).

Nesterov Accelerated Gradient performs the best with the right parameters. The optimal point is reached faster and in a smoother way. The importance of the parameters lies in the choice of the time-step because in general, a big time-step will induce divergence. This will be further explained in 4.2.

We also tried to detect the optimization effect of using Nesterov instead of the classical Gradient Descent in another application within an area that is benefiting from the improvements of optimization methods, especially first order methods, which is deep learning. In this example, we took a simple neural network architecture that will be trained on the MNIST dataset[5]. The architecture is shown in figure 5.

Considering that the number of iterations in this example is fixed, we will evaluate for the same time span the accuracy of the model, i.e how close were the parameters to the optimal values using both Nesterov and the classical Gradient Descent.

We tried different network architectures by varying the size of the hidden layers. The accuracy is plotted with respect to the size of the parameters in the neural network which depends on the size of the hidden layers. The number of parameters vary from the order of 10^4 to the order of 10^5. The performance

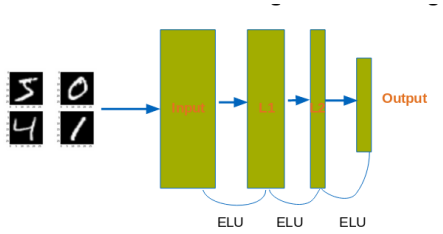


Figure 5: The Neural Network Architecture

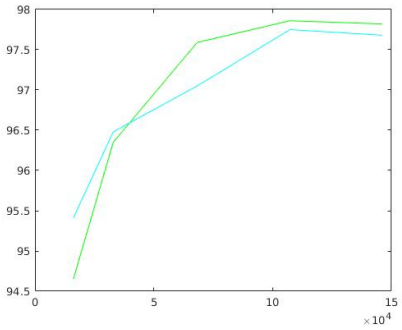


Figure 6: Comparison of Nesterov and Vanilla Gradient Descent for Neural Network Training

of both methods are plotted against each other in figure 6 where green stands for the Nesterov Accelerated Gradient and blue stands for the Gradient Descent.

It can clearly be noticed that Nesterov outperforms the classical Gradient Descent in most of the cases, especially for bigger neural networks architectures.

4 Insights into the Behaviour of The Nesterov Accelerated Gradient

Through this section, we will use results advanced in [2] to explain the behaviour of the Nesterov Accelerated Gradient. This is mainly based on a second order differential equation. So, we will first start by deriving this differential equation and then we will use it to obtain results concerning the convergence rate and other properties.

4.1 Derivation of a Second Order Differential Equation for the Nesterov Scheme

To derive the ODE, we will use the hint consisting of the three time spans $k-1$, k and $k+1$ that portray a second order differential equation, because from these three data points we can obtain the velocity and the acceleration.
We will use the update rules as described in [2] similar to the update rules mentioned in 2.2.

$$\begin{cases} x_k = y_{k-1} - s * \nabla f(y_{k-1}) \\ y_k = x_k - \frac{k-1}{k+2}(x_k - x_{k-1}) \end{cases}$$

We then combine these two rules as follows:

$$\begin{aligned} x_{k+1} &= y_k - s * \nabla f(y_k) \\ &= x_k + \frac{k-1}{k+2} * (x_k - x_{k-1}) - s * \nabla f(y_k) \end{aligned}$$

From this we deduce:

$$\frac{x_{k+1} - x_k}{\sqrt{s}} = \frac{k-1}{k+2}\frac{x_k - x_{k-1}}{\sqrt{s}} - \sqrt{s} * \nabla f(y_k)$$

We introduce the following notations:

$$\begin{cases} t = k\sqrt{s} \\ X(t = k\sqrt{s}) = x_k \\ X(t + \sqrt{(}s)) = x_{k+1} \end{cases}$$

We now use Taylor expansion on the last equation:

$$\begin{cases} \frac{x_{k+1} - x_k}{\sqrt{s}} = \dot{X}(t) + \frac{1}{2}\ddot{X}(t)\sqrt{s} + o(\sqrt{s}) \\ \frac{x_k - x_{k-1}}{\sqrt{s}} = \dot{X}(t) - \frac{1}{2}\ddot{X}(t)\sqrt{s} + o(\sqrt{s}) \end{cases}$$

For $\nabla f(y_k)$, we are using a heuristic and we are supposing that through a Taylor expansion we obtain $\sqrt{s}\nabla f(y_k) = \sqrt{s}\nabla f(X_t) + o(\sqrt{s})$

Taking all of these considerations into account we obtain:

$$\dot{X}(t) + \frac{1}{2}\ddot{X}(t)\sqrt{s} + o(\sqrt{s}) = (1 - \frac{3\sqrt{s}}{t})(\dot{X}(t) - \frac{1}{2}\ddot{X}(t)\sqrt{s} + o(\sqrt{s})) - \sqrt{s}\nabla f(X_t) + o(\sqrt{s})$$

This is equivalent to:

$$\sqrt{s}(\ddot{X}(t) + \tfrac{3}{t}\dot{X}(t) + \nabla f(X_t) - \tfrac{3}{2t}s\ddot{X}(t) + o(\sqrt{s}) = 0$$

We then divide by $\sqrt{s}$ to obtain:

$$\begin{aligned} J &= (\ddot{X}(t) + \frac{3}{t}\dot{X}(t) + \nabla f(X_t) - \frac{3}{2t}\sqrt{s}\ddot{X}(t) + o(1) \\ &= 0 \end{aligned}$$

When $s \to 0$, we obtain:

$$\lim_{s\to 0} J = \ddot{X}(t) + \tfrac{3}{t}\dot{X}(t) + \nabla f(X(t))$$

The differential equation is then:

$$\boxed{\ddot{X}(t) + \frac{3}{t}\dot{X}(t) + \nabla f(X(t))} \tag{1}$$

4.2 Convergence Rate

The factor $\frac{3}{t}$ in the ODE is the friction factor and it explains the behaviour of the convergence for the Nesterov method. In fact, one of the conventions was that $t = k\sqrt{s}$, where s is the step size. The behaviour can be summarized below and it confirms that a very large step induces divergence and overshooting:

t small	t large
$\frac{3}{t}$ large	$\frac{3}{t}$ small
Overdamped System	Underdamped system
Equilibrium reached fast	Overshooting and Divergence Risk

Table 1: Effect of the friction parameter

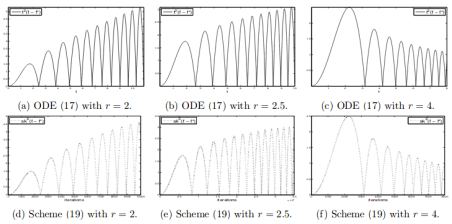
In figure 7, the effect of the variation in the friction parameter on the behaviour of the convergence is shown to portray the properties summarized in the table 1.

Now, we want to prove the convergence rate $O(\frac{1}{t^2})$ of the Nesterov scheme using the ODE. This means that:

$$f(X(t)) - f^* \leq 2\frac{||x_0 - x^*||^2}{t^2}$$

For this purpose, we consider the following energy functional and its derivative that are used to study the system's equilibrium:

$$\begin{cases} \epsilon(t) = t^2(f(X(t)) - f^*) + 2||X + \frac{t}{2}\dot{X} - x^*||^2 \\ \dot{\epsilon}(t) = 2t(f(X) - f^*) + t^2 < \nabla f, \dot{X} > +4 < X + \frac{t}{2}\dot{X} - x^*, \frac{3}{2}\dot{X} + \frac{t}{2}\ddot{X} > \end{cases}$$



(a) ODE (17) with $r = 2$. (b) ODE (17) with $r = 2.5$. (c) ODE (17) with $r = 4$.

(d) Scheme (19) with $r = 2$. (e) Scheme (19) with $r = 2.5$. (f) Scheme (19) with $r = 4$.

Figure 7: Effect of the choice of the friction parameter on the convergence properties[2]

From the ODE, we have:

$$\frac{t}{2}\ddot{X}(t) + \frac{3}{2}\dot{X}(t) = -\frac{t}{2}\nabla f(X(t))$$

We replace it into $\dot{\epsilon}$ and considering that f is convex we obtain:

$$\begin{aligned}\dot{\epsilon} &= 2t(f(X) - f^*) - 2t < X - x^*, \nabla f(X) > \\ &\leq 0\end{aligned}$$

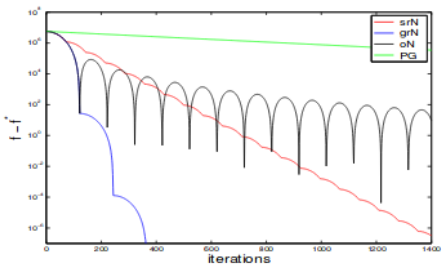
We deduce that ϵ is decreasing and from that follows the following:

$$\begin{aligned}f(X(t)) - f^* &\leq \frac{\epsilon(t)}{t^2} \\ &\leq \frac{\epsilon(0)}{t^2} \\ &= 2\frac{||x_0 - x^*||^2}{t^2}\end{aligned}$$

We then obtain the result confirming that the convergence rate is of $O(\frac{1}{t^2})$ using the ODE.

4.3 Strong Convexity

Strong convexity is a desirable property for functions as it allows a linear convergence rate using the classical Gradient Descent algorithm. However, in Nesterov the convergence rate is $O(\frac{1}{t^2})$ or might even be $O(\frac{1}{t^3})$.
A way to overcome this problem is by using **Restarting**, which resets the friction factor, because as the time evolves the friction factor decreases and as a result the system will exhibit overshooting which explains why a lot of oscillations occur.



(a) $\min \frac{1}{2}x^T Ax + bx.$

Figure 8: Effect of restarting on the convergence rate[2]

A value that can be chosen for the restarting of the friction factor is $\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$ where L is the Lipschitz-coefficient of the function.
We can observe the effect of restarting in the figure 8 below which shows that from the oscillatory behaviour of the Nesterov method (black) we can switch to a linear convergence rate (red) if we use the Restarting method.

5 Conclusion

In the present review, we have seen the need for a well-performing first order method for optimization problems. Specifically, we have seen the Nesterov Accelerated Gradient and its characteristics.

We also touched on some of the important points that were used to describe the Nesterov scheme and explain the underlying behaviour, notably the convergence rate and the method of restarting for strongly convex functions used to ensure a convergence rate comparable with the classical Gradient Descent. These explanations were advanced thanks to the usage of a second order differential equation.

List of Figures

List of Tables

References

[1] A. Karpathy. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/neural-networks-3/#sgd.

[2] W. Su, S. Boyd, and E. Candès. "A differential equation for modeling Nesterov's accelerated gradient method: Theory and insights". In: 17 (Sept. 2016).

[3] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. "Advances in Optimizing Recurrent Networks". In: *Acoustics, Speech, and Signal Processing, 1988. ICASSP-88., 1988 International Conference on* (Dec. 2012). DOI: 10.1109/ICASSP.2013.6639349.

[4] ZhouYuxuanYX. *Matlab-Implementation-of-Nesterov-s-Accelerated-Gradient-Method-*. URL: https://github.com/ZhouYuxuanYX/Matlab-Implementation-of-Nesterov-s-Accelerated-Gradient-Method.

[5] J. Langelaar. *MNIST neural network training and testing*. URL: https://www.mathworks.com/matlabcentral/fileexchange/73010-mnist-neural-network-training-and-testing.