MOMENTUM AND NESTEROV'S ACCELERATED GRADIEND METHODS

Franz Franco Gallo, Minh Nhat Do

Université Côte d'Azur, 06300 Nice, France

ABSTRACT

Momentum method is a technique for accelerating the stochastic gradient descent(SGD) method, the momentum update can be motivated from a physical perspective by introducing a velocity component. A velocity vector is accumulated in directions that consistenly reduce the cost function, allowing update the parameters in a direction that is more effective than steepest descent. Nesterov's accelerated gradient(NAG) is a variant of momentum method, the main difference is that we also apply a momentum update to the parameters and the gradient is computed with these updated parameters in each step. Momentum methods approach most of the time achieve better convergence rates on deep neural networks compared with vanilla SGD.

1 INTRODUCTION

1.1 MOMENTUM METHOD

The Momentum(Qian, 1999) upgrade can be motivated from a physical perspective of the optimization problem. In particular, the loss can be interpreted as the height of mountainous terrain and therefore also as the potential energy U. Initializing the parameters with random numbers is equivalent to setting a particle with zero initial velocity somewhere in this terrain. The optimization process can then be seen as equivalent to the process of simulating a particle rolling across the landscape.

Since the force on the particle is related to the potential energy gradient $F = -\nabla U$, the force felt by the particle is the negative gradient of the loss function. Also, $F = ma$, so the negative gradient is proportional to the acceleration of the particle. The physics view suggests an update where the gradient directly influences speed, which is different from stochastic gradient descent where the gradient directly integrates the position.(Ruder, 2016)

The moment method accumulates a velocity vector in directions of persistent reduction of the cost function. This property allows momentum accumulate velocity in low curvature directions to generate accelerated progress in such directions compared to SGD. Standard momentum is given by the following pair of equations:

$$v_{t+1} = \mu v_t - \alpha \nabla f(\theta_t) \tag{1}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{2}$$

Where μ is the momentum parameter or in contrast $(1 - \mu)$ is the friction parameter, the variable $\alpha > 0$ represent the learning rate and $\nabla f(\theta_t)$ is the the unbiased estimate of the gradient at θ_t. The momentum constant μ controls the decay of the velocity vector v, which values closer to 1 result in the gradient information persisting over more iterations, which generally leads to higher velocities.

1.2 NESTEROV'S ACCELERATED GRADIENT

Nesterov's Momentum is a slightly different version of the momentum update and achieves stronger theoretical convergence for convex functions that performs better than standard momentum.

The core idea behind Nesterov's momentum is that when the current parameter vector is at any position θ_t, then looking at the previous momentum update, we know that the momentum term is

about to push the vector parameter by μv_t. Therefore, if we are about to calculate the gradient, we can treat the future approximate position θ_t + μv_t as an "lookahead", this is a point in the neighborhood of where we will soon going to end up. So it makes sense to compute the gradient at θ_t + μv_t rather than at the old position θ_t. The process Nesterov's Momentum momentum is given by the following pair of equations:

$$v_{t+1} = \mu v_t - \alpha \nabla f(\theta_t + \mu v_t) \tag{3}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \tag{4}$$

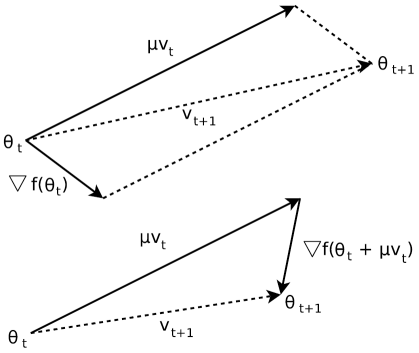


Figure 1: One step on Momentum method(top) and Nesterov's accelerated gradient(bottom).(Sutskever et al., 2013)

The key difference between Momentum and Nesterov's accelerated gradient is that Nesterov's momentum instead of computing the gradient at the current position θ_t, it computes the gradient after the velocity effect in a "lookahead" position $\theta_t + \mu v_t$ (Figure 1).

2 IMPLEMENTATION

2.1 TOY EXAMPLE: LINEAR REGRESSION

Synthetic data is created around a line by adding noise to the output. So, effectively, :

$$y = ax + b + \varepsilon, \quad where\, \varepsilon \in N(0, \sigma^2)$$

The data is a line with some gaussian noise added, where the parameters of the ground truth are known: a = -3 and b = 10 . The mean squared error is defined as follows:

$$F = \frac{1}{n} \sum_{i=1}^{n} (y_i - (ax_i + b))^2$$

To compute the Momentum method according to the equations (1) and (2) in section 1.1, the variable velocity v_t is initialized to zero and the hyperparameter μ known as momentum is set to a typical value 0.9, the number of epochs is set to 100, and the learning rate is $5 \cdot 10^5$. The parameters a and b can be initialized randomly but for our purposes we initialized the value of a = 8 and b = 75. So we define our parameter $\theta_1 = [a, b]$. For Nesterov's implementation we use equation (3) and (4) with the same initializations of the parameters in Momentum method so we can compare the results later.

In Figure 2 we can see the Momentum updates in the parameters a and b, this convergence is faster than Mini Batch Gradient Descent with the same hyper parameters, however we can realized that there is a lot of noise especially for the parameter a, and this is because sometimes the acceleration that momentum takes is not the optimal, and that is shown in the velocity graph that presents a lot of randomness at the beginning so adding noise. On the other hand in Figure 3, we observe Nesterov's updates considerably reduce the initial noise present in the Momemtum method, and this is expected because Nesterov takes the gradient in an "lookahead" position and corrects the direction even before being in that position. In summary from this experiment we can say that Nesterov notably increases the speed of convergence but at the end both achieve almost the same results, for train deep neural networks this small change in Nesterov will make a big difference on the convergence velocity for these models with thousands of parameters.(Goodfellow et al., 2016)

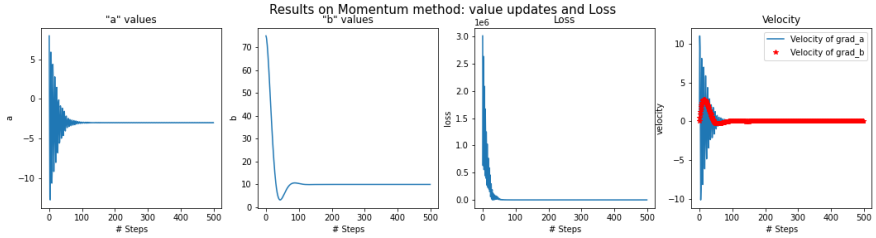


Figure 2: Momentum updates on parameters a, b, loss and velocity respectively

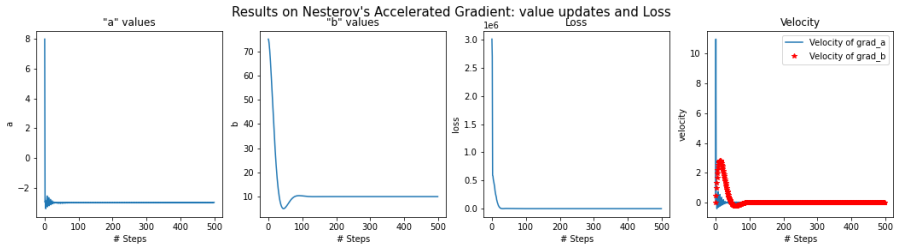


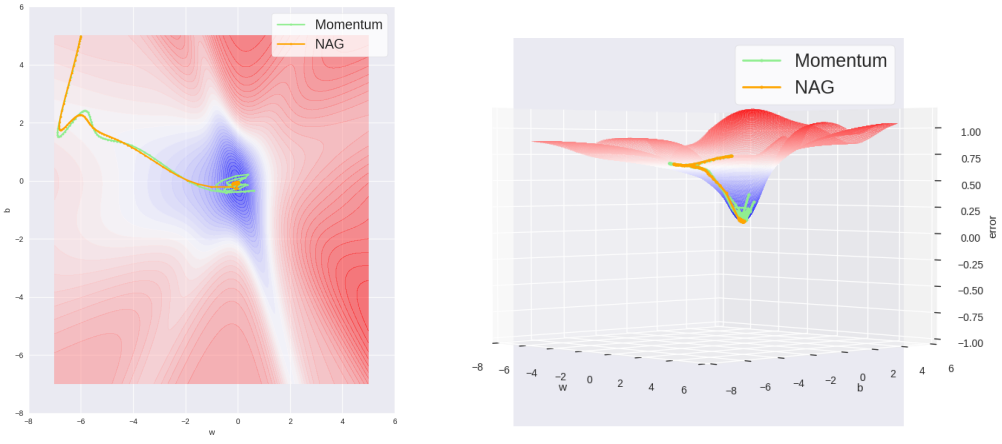
Figure 3: Nesterov's updates on parameters a, b, loss and velocity respectively

2.2 TOY EXAMPLE SIGMOID FUNCTION

For this toy example we use the same process of initialization in section 2.1. To implement the gradient descent optimization technique, in this example we will take a simple case of the perceptron with a sigmoid activation function (sigmoid neuron) with the parameters w - weight and b - bias. We have the formula of Sigmoid function as follows:

$$y_{w,b}(x) = \frac{1}{1 + e^{-w^T x + b}}$$

This example will show how different Momentum method and Nesterov's Accelerated Gradient Method learn the parameters w and b of function Sigmoid. We will create a $Sigmoid$ class and then create 2 inheritance classes, Momentum and NAG. Then we create 2 models from the above 2 classes representing the 2 methods to be compared during learning and updating both parameters w and b. The $Sigmoid$ class will provide functions on calculating gradient on w, b, loss values. The Momentum class and NAG class provide different fit functions to learn from data.



(a) 2D Surf face learning process

(b) 3D animation visualization

Figure 4: Visualization of gradient history on 2D and 3D graph

In the visualization, the objective of the learning algorithm is to move the particle towards the dark blue color region where the error/loss is minimum. To have a multi-dimensional viewing angle, we have built two display methods: 3D and 2D graph animations. In this case we notice something slightly different from the toy example in regression, because in Figure 4 at the beginning both models have the same behavior but when they are reaching the confusion region the Momentum method starts jumping in random directions and this effect might be due to the value of the learning rate, because it starts in a smooth area and then it goes in to steepest part of the function to reach the minima.

2.3 NEURAL NETWORK

This example will show how different Momentum method and Nesterov's Accelerated Gradient (NAG) Method learn on a feed forward Neural network. We will create a simple Neural network with 2 layers. We used a softmax function as a activation function to run this example. We will compare based on accuracy and loss value on each applied method. We have added the SGD(Ruder, 2016) to compare with Momentum and Nesterov's method.

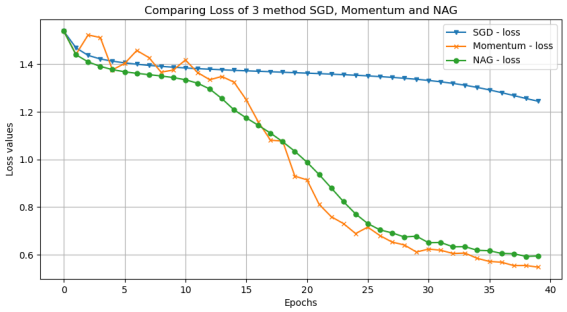


Figure 5: Loss comparison of methods SGD, Momentum and NAG

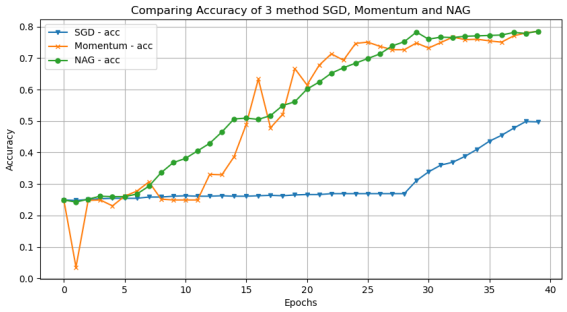


Figure 6: Accuracy comparison of methods SGD, Momentum and NAG

With SGD we don't compute the exact derivate of our loss function. Instead, we're estimating it on a small batch. Which means we're not always going in the optimal direction, because our derivatives are noisy. Just like in the graphs above. As it is expected the Nesterov method have a smooth converge in comparison with Momentum.

In Figure 5, we plot the loss values, the performances of Momentum and NAG are almost the same and NAG is slightly better. Unlike Momentum, NAG has smoother steps and does not form peaks. SGD is the slowest and easy to be trapped in local minimum as the plot of loss is shown. After training for 100 epochs, both Momentum and NAG reach 79% but SGD is still trapped in 50%.(Figure 6)

3 CONCLUSION

When the learning rate is relatively large, Nesterov Accelerated Gradients allows larger decay rate than Momentum method, while preventing oscillations. However both Momentum method and Nesterov Accelerated Gradient become equivalent when the learning rate is small.

Stochastic gradient descent does not exactly provide us with the optimal direction in which the loss function is headed, the steepest descent. Therefore, we might not always be headed in the optimal direction, this is primarily because the earlier derivatives of the loss function act as a noise in the later stages of the updating the weights. This causes slow training and convergence. Momentum technique helps us solve this issue of slow convergence.

REFERENCES

Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL http://www.deeplearningbook.org/.

Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12 (1):145–151, 1999. doi: 10.1016/S0893-6080(98)00116-6. URL https://doi.org/10.1016/S0893-6080(98)00116-6.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pp. 1139–1147. JMLR.org, 2013. URL http://proceedings.mlr.press/v28/sutskever13.html.

A APPENDIX

.1 GITHUB PROJECT LINK