Momentum Based Optimization Techniques on 2 Scales

Tara Braswell

December 2025

1 Abstract

Momentum based methods, in particular ADAM, have become standard practice in Machine Learning instead of pure Steepest descent. In this paper, we will discuss the conceptual motivations behind these methods, and show the relative effectiveness on two toy problems. We will begin with an implementation on the Rosenbrock function for a baseline in small problems (2 variables), before moving on to a machine learning problem with several hundred variables.

2 Introduction

Hessians are expensive, and they do not scale well. While Newton's method provides an extremely straightforward and efficient algorithm for minimization, the storage requirements for the Hessian scales quadratically with the number of variables, and inversion takes $\mathcal{O}(n^3)$ steps. For machine learning, with several million hidden variables to optimize, this is unfeasible, but gradient descent can be improved on to capture higher order behavior.
Enter Momentum based methods. Initially described by Polyak's "heavy ball" method for numerical solutions to differential equations, Momentum based methods attempt to use previous values of $\bar{p_k}$ to capture any details that are overlooked by steepest descent/gradient descent*Stochastic Gradient Descent* n.d. The initial formulation of Polyak's heavy ball is a simple modification of of steepest descent:

$$\bar{p}_k^{HB} = (\beta)\bar{p}_{k-1}^{HB} + (1-\beta)(-\nabla f(\bar{x_k})), \beta \in (0,1].$$

This idea of convex combination of steepest descent with previous steps was noticed in 1986 by Rummelhart, Hinton and Williams, who were explicitly looking for Hessian-free methods for simple neural netsRumelhart, Hinton, and Williams 1986.
ADAM,or ADAptive Momentum estimation, combines this idea of "momentum" or a delay term in the step direction with rescaling each vector term by an exponentially decaying (root-mean-square) average of its previous value. By

rescaling the $\bar{x}_i$ component, we can "accelerate" through long flat regions by lengthening out step size in that direction. the result is an update step as follows:

$$\bar{x}_{k+1} = x_k + \eta \text{ diag}(\big[(\sqrt{v_{1,k}} + \epsilon)^{-1} \quad (\sqrt{v_{1,k}} + \epsilon)^{-1}) \dots (\sqrt{v_{1,k}} + \epsilon)^{-1}\big])\bar{p}_k^{HB}$$

where

$$v_{i,k} = \beta_2 v_{i,k-1} + (1 - \beta_2)(\bar{p}_{i,k})^2$$

. This idea of root-mean-square velocity rescaling was first mentioned by Hinton in 2012, ADAM was developed in 2015Kingma and Ba 2015, and by today, it has become the standard optimization algorithm in machine learning. While explicit convergence rates have been hard to calculate and theoretical convergence rates being at worst sublinear, empirical results have found it extremely efficient for neural net techniques.

3 Methods/Theory

3.1 Rosenbrock

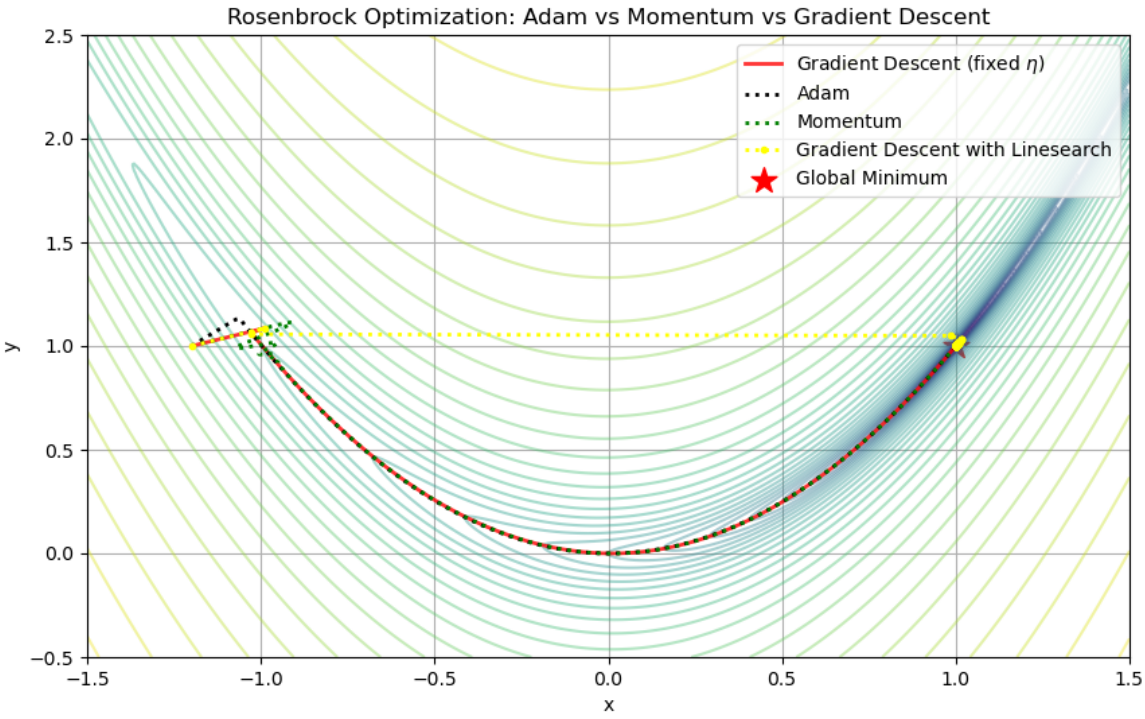
We will consider three and a half different optimization techniques on our Rosenbrock function, $f(\bar{x}) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$ starting from $\bar{x} = [-1, 1]^T$. Using fixed step size $\eta = 10^{-3}$, we will compare gradient descent, Polyak's heavy ball method ($\beta = 0.9$), and ADAM ($\bar{\beta} = [0.9, 0.99]^T$). We will also calculate a gradient descent with line searching $\rho = 1/2, a = 1$ as a reference. To analyze results, we will show the paths traveled, the size of the gradient over time, and will also compute the run time for each particular loop (As ran on 4-core Intel i5-4300U processer at 2.90 GHz on a 2013 Lenovo Thinkpad 440s). We will do this by using the %timeit function in Jupyter.

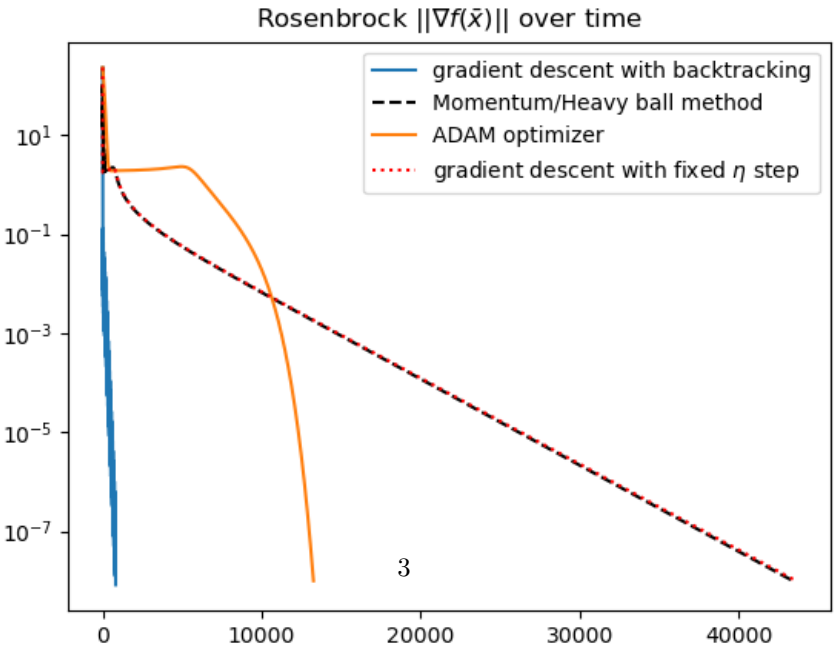
3.2 Machine Learning on MNIST

To give a more realistic demonstration, we will also consider the performance of both (stochastic) Gradient Descent and ADAM on a small neural network trained to recognize digits from the MNIST dataset. Using Pytorch, we have built a 2 layer feed-forward neural network with 784 inputs(for image size), 128 hidden neurons, and 10 outputs (likelihood of each digit), using cross-entropy loss. These will be ran for 3 epochs before testing. We will be considering a pure Stochastic gradient descent, stochastic gradient descent with momentum (heavy ball), Root-mean-square velocity,and ADAM. We will also use the %%timeit magic in Jupyter for an approximation of time, and look at the average values of both the training loss and the test accuracy at the end of each epoch. These will be ran on the same antique laptop; notably, without access to GPU.
A theoretical caveat should be made. Stochastic gradient Descent does not converge linearly, but instead sublinearly ($\mathcal{O}(\sqrt{n})$, as does ADAM.

4 Results

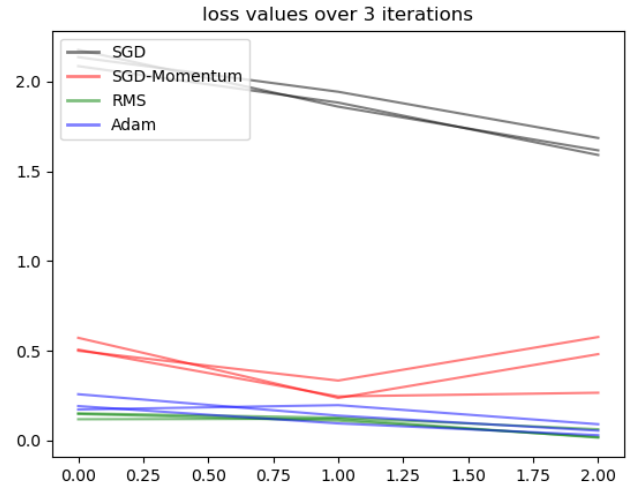
4.1 Rosenbrock

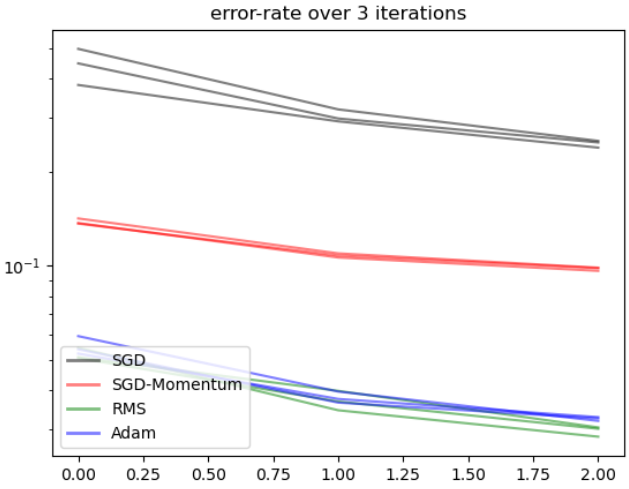




When ran, fixed step gradient descent takes an average of 630 ms, and the Heavy ball method takes 805 ms. ADAM takes an average of 347 ms, and the gradient descent with Linesearch takes 113ms. Gradient descent takes 43606 steps to converge, heavy ball method takes 43411 steps, ADAM takes 13260, and the line searching gradient descent takes 818 steps. Note that momentum and heavy ball both graph linearly (due to linear convergence rate); despite no guarantee of superlinear or even linear convergence, adam gives a roughly quadratic shape.

4.2 MNIST





Regardless of optimization algorithm, each 3-epoch train/test cycle took approximately 50 sec-

onds. While accuracy data is presented on a log-error scale for sensitivity around effective models, pure gradient descent converged to roughly 75% accuracy, momentum converges to approximately 90% accuracy

5 Discussion

Momentum based methods are not particularly suited for the Rosenbrock function, due to the problem's low dimensionality. Compared to using gradient descent with a fixed step size, heavy ball methods take slightly few steps (though more computational time) and ADAM roughly quarters the iteration count. However, all these methods are significantly slower than an effective line search using gradient descent (which for a 2-dimensional problem, still converges slower than Newton's Method). The gains from just the heavy ball method are small but noticeable in terms of step count, though the 3 additional vector operations do increase computational time for each loop. Visually, we can see an "oscillatory" pattern of the heavy ball method overshooting the bottom of our Rosenbrock valley before correcting itself. It should be noted that the convergence of ADAM is not guaranteed Reddi, Kale, and Kumar 2018, but it still performs better than the other raw gradient descent.

Momentum based techniques begin to shine in a machine learning contexts, where the total number of values to be optimized is significantly higher (in this case, 101632 individual tensor weights). Immediately, the "free" benefit of using momentum over raw gradient descent is clear; our error rate decreases from 25% to 10% with very little additional effort. Root-mean-square velocity and ADAM both cut this error rate roughly in half - ADAM performs slightly worse, though this may not be true for other problems or when we spend longer training our data.

6 Conclusion

Within traditional optimization techniques, there is a tradeoff between the convergence speed gained via computation and inversion of the Hessian, and the computational cost associated. Momentum-based methods allow us to capture some higher-order details without the need for engaging with these higher derivatives. The result gives us a method that has few advantages when the number of variables is small and the hessian is trivial to compute, but as the problem scales, we have a method that works better than gradient descent but does not require any additional storage.

References

Kingma, Diederik P. and Jimmy Ba (2015). "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. URL: https://arxiv.org/abs/1412.6980.

Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar (2018). "On the Convergence of Adam and Beyond". In.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning Representations by Back-propagating Errors". In: *Nature*.

Stochastic Gradient Descent (n.d.). https://en.wikipedia.org/wiki/Stochastic_gradient_descent. Accessed: 2025-12-12.