Learning the learning rate: modifying WNGrad

Marcus Gruneau, Chris Crisictiello, Alejandro Noguerón
CS-439 mini-project
Due June 18, 2021

***Abstract*—In this report we study a simple modification of** SGD **proposed in [1], named** WNGrad**. This modification does not require learning-rate tuning, as most gradient methods do. Furthermore, we present a modified version of** WNGrad**, called** $\mathrm{WNGrad}-\mathrm{mod}$**, which makes the learning rate decay more slowly, in some cases speeding up optimization and improving learning. We compare** $\mathrm{WNGrad}-\mathrm{mod}$ **with well established algorithms as** SGD**,** Adam**, line-search methods (when applicable) and** WNGrad**. To get an overall idea of the behaviour, we consider convex problems (linear regression, SVM), high-dimensional and nonconvex problems (deep neural networks), and optimization under a manifold constraint (Rayleigh quotient, robust principal subspace analysis). For the latter class of problems we use the straightforward extension of** WNGrad **and** $\mathrm{WNGrad}-\mathrm{mod}$ **to the Riemannian setting. Finally, we will evaluate the parameter initialization of** $\mathrm{WNGrad}-\mathrm{mod}$ **and** WNGrad **proposed in [1].**

I. Introduction

Setting the learning rate for optimization methods in ML such as gradient descent often requires manual tuning to achieve the best results. Moreover, it is well known that a big learning rate will oscillate around the solution, while a very small learning rate will take too long to find it. Meta-learning is a class of methods which automate hyper-parameter tuning by gradually learning or adjusting the parameter during the optimization process. Some well known example algorithms are Adam [2] and Adagrad [3]. Although these algorithms require an initial hyperparameter, they are less sensitive than SGD to the choice of the hyperparameter.

To get rid of this hyperparameter, we implement an algorithm from the recent paper [1] in which the authors propose the following update rule termed WNGrad:

$$\begin{aligned} x_{k+1} &= x_k - \frac{1}{b_k}\nabla f(x_k) \\ b_{k+1} &= b_k + \frac{1}{b_k}\|\nabla f(x_{k+1})\|^2. \end{aligned} \tag{1}$$

In (1), $\frac{1}{b_k}$ is the learning rate. In order to initialize the first iteration of the algorithm, one needs to choose b_0. To automate this process [1] suggests sampling the gradient at points $\{u_j\}$ near x_0, and setting

$$b_0 = \max_j \frac{\|\nabla f(u_j) - \nabla f(x_0)\|}{\|u_j - x_0\|} \leq L, \tag{2}$$

Chosen in this way, b_0 provide a lower bound on the true smoothness parameter L. If this is not possible or not feasible, then one can also initialize b_0 by using the ℓ_2 norm of the gradient evaluated at x_0 multiplied by a constant $C \geq 0$.

We investigate and evaluate this algorithm by applying it to different types of optimization problems (convex and non-convex) and compare the results. For comparison and benchmarking purposes, we will test against SGD with a manually tuned learning rate.

Finally, we present the following simple modification to the update rule (1)

$$\begin{aligned} x_{k+1} &= x_k - \frac{1}{b_k}\nabla f(x_k) \\ b_{k+1} &= b_k + \frac{1}{b_k}\|\nabla f(x_{k+1})\|, \end{aligned} \tag{3}$$

which we will call $\mathrm{WNGrad}-\mathrm{mod}$. Note the update of b_{k+1} now uses the gradient norm rather than the squared gradient norm. This modification is inspired by observations that WNGrad often failed to reach objective function values as low as SGD, usually because b_k would grow too large too fast, and thus the learning rate would become too small. As we will present in section II, we found this modification to have favorable results in some cases.

II. Experiments

In this section we present numerical experiments to illustrate the convergence properties of the our proposed modification of the WNGrad. We will present comparisons to the original WNgrad proposed in [1], as well as to other standard algorithms.

A. Convex Setting

1) Linear Regression: We set up our first experiment using the data set from lab 2, where the goal is to minimize the sum of squares

$$f(x) = \frac{1}{2n}\sum_{i=1}^{n}(a_i^\top x - b_i)^2 = \frac{1}{2n}\|Ax - b\|^2. \tag{4}$$

We chose this setting as a first, simple test for WNGrad because the loss function is convex and the complexity of the data set allows us to run full gradient descent. Moreover, the Lipschitz constant L is known, which allows us to see the behaviour of WNGrad when b_0 is larger, smaller or similar to the Lipschitz constant. To investigate this, we pick some arbitrary learning rates and compare them to the performance of WNGrad.

We start out by using $C\nabla f(x_0)$ as an approximation for b_0. As can be seen in Figure 1, the initial value of C affects the size of the first step and the rate of convergence. With a higher value of C, the gradient is scaled up by a large factor, resulting

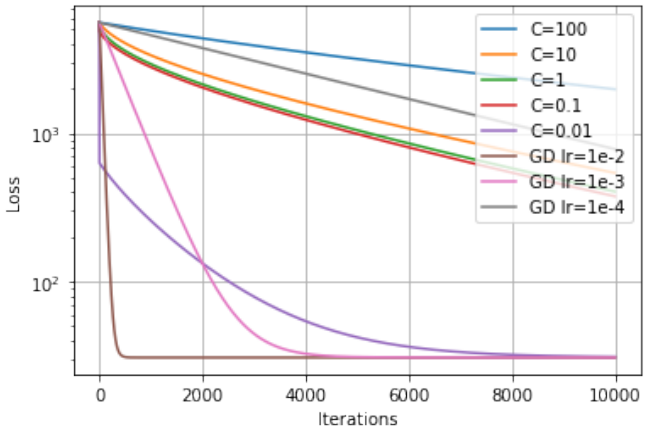

Fig. 1. WNGrad vs. Gradient descent.

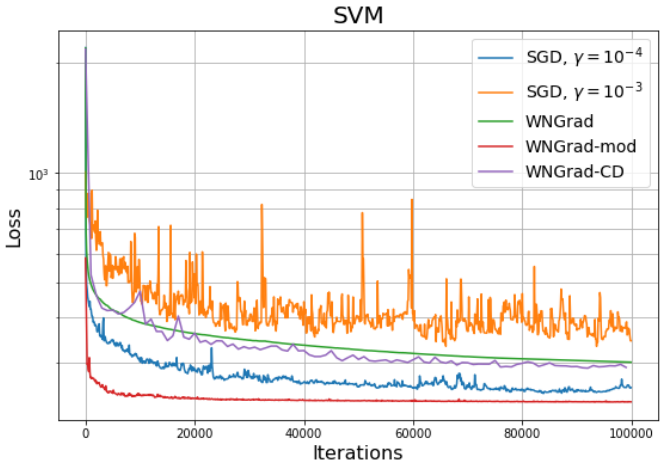

Fig. 2. Results of minimizing the SVM objective with SGD with $\gamma = 10^{-4}$, 10^{-3}, WNGrad and WNGrad, all with a mini-batch size of 30. The random sampling was done by adding gaussian noise sampled from the distribution $\mathcal{N}(0, 10^t)$, for $t \in [-3, 3]$.

in a small initial step size and slow descent. By lowering the value of C the initial gradient step is much larger, and the learning rate seems to stabilize albeit slower than normal gradient descent with a step size closer to $\frac{1}{L}$. Selecting any value smaller than 10^{-4} for this setting causes the gradient to explode. From this results, it is clear that b_0 is a delicate parameter, and C an important hyperparameter. To get rid of this hyperparameter, in the rest of the experiments we use the initialization rule (2).

2) SVM: We minimize the standard SVM objective applied to the dataset `w1A` presented in [4], parting from lab 9. We try SGD with two different values of the stepsize γ (10^{-4}, 10^{-3}), WNGrad and WNGrad $-$ mod, both with a batch size of 30.

As Figure 2 shows, WNGrad has a slow convergence, and the parameter b becomes too large too fast for the algorithm to converge to the optimal value. Nonetheless, it is still noticeable how it always reaches good results without the need for any hyperparameters, unlike SGD.

On the contrary, WNGrad $-$ mod has the same initialization of b as WNGrad, but b grows at a slower rate, meaning that γ decreases at a slower rate, reaching convergence even faster than SGD. This is because for the first few iterations, γ is bigger than 10^{-4}, and then settles down at an optimal value.

Finally, we also implemented a version of coordinate descent

$$x_{k+1}^{(i)} = x_k^{(i)} - \frac{1}{b_k}(\nabla f(x_k))^{(i)}, \quad b_{k+1} = b_k + \frac{1}{b_k}[(\nabla f(x_k))^{(i)}]^2,$$

where of course $x_{k+1}^{(j)} = x_k^{(j)}$ for all coordinates $j \neq i$. The resulting method, called WNGrad-CD, performs the same as WNGrad.

This example shows how well WNGrad $-$ mod works in the convex case. It can sample a wide space and get a good estimate of L, and after several iterations b_k becomes large enough for the optimization to stabilize.

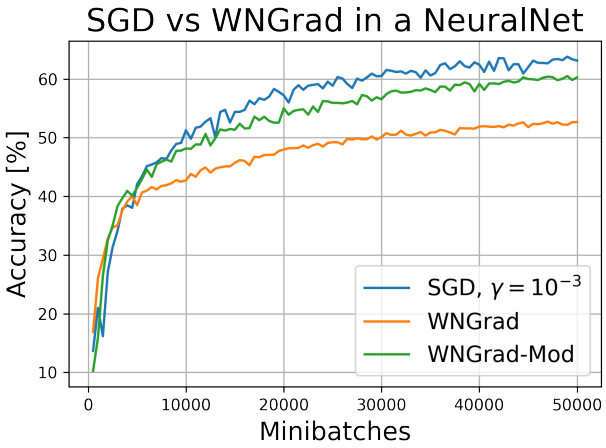

Fig. 3. Results of CIFAR10 Classification with 6 layer neural network. 50000 minibatches -thus parameter updates- were done, which correspond to 10 epochs. Experiments were done with the same architecture, using the standard optimization algorithms SGD ($\gamma = 10^{-3}$, blue line), as well as WNGrad (yellow line) and WNGrad $-$ mod (green line) with automatic b_0 initialization.

B. High-dimensional nonconvex setting

1) Convolutaional Neural Network: For more high-dimensional, nonconvex settings, we tested WNGrad $-$ mod on the CIFAR10 dataset, using a small network consisting of 2 convolutional layers (with one max-pooling layer in the middle) followed by 3 fully connected layers. The model, parameters, and implementation are modifications of this tutorial [5] from the official PyTorch documentation. The optimizers compared are SGD (with the parameters proposed in [5]), and WNGrad and WNGrad $-$ mod with automatic initialization of b_0.

As we can see in Figure 3, $\mathrm{WNGrad-mod}$ is slightly outperformed by SGD after 10 epochs. We attribute this to the constant growth of b_k, which slows down optimization after a few epochs. This effect is even more pronounced for WNGrad, which, after the first few hundred minibatches, achieves the highest accuracy, but then slows down and eventually is outperformed by the rest of the algorithms. While a properly tuned SGD is still clearly better in non-convex settings, it is still remarkable how $\mathrm{SGD-mod}$ can achieve almost the same accuracy without any hyperparameter tuning.

C. Riemannian setting

We extend WNGrad to optimization on Riemannian manifolds in the natural way: replace the Euclidean gradient with the Riemannian gradient – see [6] for an introduction to Riemannian optimization. Similarly, initialization is also performed in the natural way: estimate b_0 using

$$b_0 = \max_j \|\nabla f(u_j) - T_{x_0 \to u_j} \nabla f(x_0)\| / \mathrm{dist}(x, u_j)$$

where T_{u_j, x_0} is any transporter [6] from x_0 to u_j, preferably one that is close to parallel transport.

We then test the performance of WNGrad on two problems.

1) Robust principal subspace analysis: We are given $m = 2000$ points $z_1, ..., z_m \in \mathbb{R}^n = \mathbb{R}^{200}$ which we lie close to an unknown subspace S of dimension $p = 10$. We also assume that a proportion $q = 30\%$ of the points are outliers which lie close to a different p-dimensional subspace. The deviation of the points from the subspaces is controlled by a variance parameter σ^2. The goal is to recover S. The optimization problem takes the form

$$\min_{X \in \mathrm{Gr}(n,p)} f(X), \quad \text{with} \quad f(X) = \sum_{i=1}^{m} \sqrt{||z_i - X z_i||^2 + \epsilon^2}.$$

where the Grassmann manifold $\mathrm{Gr}(n, p)$ is the set of p-dimensional subspaces of $\mathbb{R}^n$. For computational purposes, we represent $\mathrm{Gr}(n, p)$ as a smooth manifold of orthogonal projection matrices [7]. Note that the cost function f has a finite sum structure and uses the so-called pseudo-Huber loss.

In Figure 4, we compare SGD, WNGrad, $\mathrm{WNGrad-mod}$, Riemannian gradient descent (RGD) with line search and full gradients, and the Riemannian trust region method (RTR) [8] with full gradients and hessian-vector products. In this case, iteration number is not a faithful representation of efficiency, so we plot the function cost vs time. We see that initially $\mathrm{WNGrad-mod}$ is significantly faster than all other methods. However, $\mathrm{WNGrad-mod}$ begins to slow prematurely and is outperformed by WNGrad, SGD, RGD and RTR. Overall WNGrad gives the best performance, converging to the same function value as RGD and RTR, while requiring significantly less computation time. This suggests that, at least for this problem, it would perhaps be beneficial to initially use the stepsize scheme due to $\mathrm{WNGrad-mod}$ and midway to switch to the scheme used by WNGrad. [1]

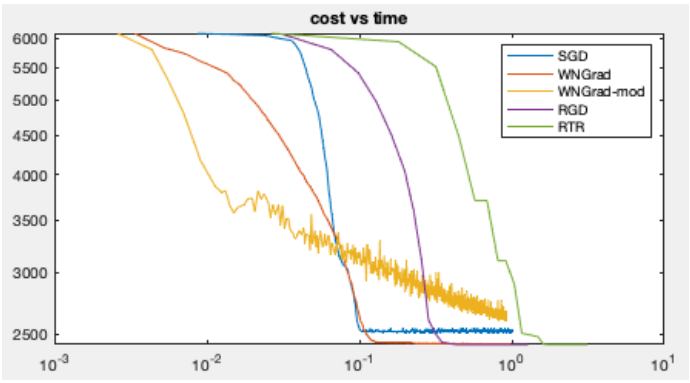

Fig. 4. Here $n = 200, m = 2000, p = 10, q = 0.3, \epsilon = 10^{-5}, \sigma^2 = 0.001$. We tried four different values when tuning the stepsize of SGD. All stochastic gradients are calculated using a minibatch size of 20, which is 10% of the data. The vertical axis of the log-log plot is the function cost, and the horizontal axis is computation time. The metric projection retraction is used in all cases.

2) Rayleigh quotient: We consider the Rayleigh quotient for finding minimal eigenvectors of a symmetric matrix A:

$$\min_{x \in \mathrm{S}^d} f(x) = x^\top A x$$

where S^d denotes the d-dimensional sphere. We consider $d = 100$. We artificially add Gaussian noise of variance σ^2 at each iteration to create stochastic gradients.

Instead of comparing SGD, WNGrad and $\mathrm{WNGrad-mod}$, which yields similar results as in the previous example, we now experiment with a slight variation on $\mathrm{WNGrad-mod}$. For different values of p, we try stepsize updates of the form

$$b_{k+1} = b_k + \frac{1}{b_k} \|\nabla f(x_{k+1})\|^{2/p} \quad (\mathrm{WNGrad-mod-p})$$

where $\nabla f(x)$ is the Riemannian gradient. Observe that when $p = 1$ this is exactly WNGrad, and when $p = 2$ this instead becomes $\mathrm{WNGrad-mod}$.

Figure 5 shows the results of running $\mathrm{WNGrad-mod-p}$ with various values of p and gradient noise σ^2. When the noise is small, we see that $\mathrm{WNGrad-mod}$ ($p = 2$) converges quickest. On the other hand, when the noise is larger, WNGrad gives the best performance.

III. Conclusion

In this report, we have shown that in several cases, the algorithm $\mathrm{WNGrad-mod}$ can outperform other standard optimization algorithms. Moreover, we have shown how in nonconvex settings $\mathrm{WNGrad-mod}$ does achieve results almost as good as most standard algorithms after fine-tuning.

[1] The code for this example is based on manopt [9] as well as code developed by Professor Nicolas Boumal and one of the authors for Project 2 of the EPFL course MATH-512. Nicolas Boumal granted permission to use this code.

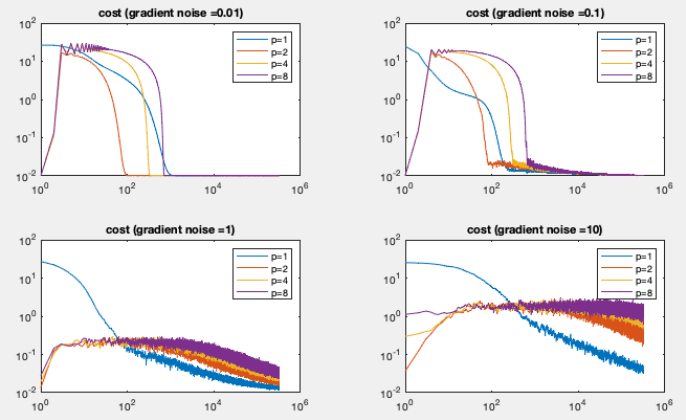

Fig. 5. Results of running WNGrad $-$ mod $-$ p for various values of p and the gradient noise (σ^2). The vertical axis of the log-log plot is the function cost, and the horizontal axis is the iteration number. (We do not include results for $p \leq 1/2$ because the performance was significantly worse than any of the other values of p shown.) The metric projection retraction is used in all cases.

REFERENCES

[1] X. Wu, R. Ward, and L. Bottou, "Wngrad: Learn the learning rate in gradient descent," 2020.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[3] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization." *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/jmlr/jmlr12.html#DuchiHS11

[4] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization." *Advances in Kernel Methods - Support Vector Learning*, 1998.

[5] "Training a classifier," PyTorch website, accessed: 2010-09-30.

[6] N. Boumal, "An introduction to optimization on smooth manifolds," Available online, Nov 2020. [Online]. Available: http://www.nicolasboumal.net/book

[7] T. Bendokat, R. Zimmermann, and P. A. Absil, "A grassmann manifold handbook: Basic geometry and computational aspects," 2020.

[8] N. Boumal, P.-A. Absil, and C. Cartis, "Global rates of convergence for nonconvex optimization on manifolds," *IMA Journal of Numerical Analysis*, vol. 39, no. 1, pp. 1–33, Feb. 2018.

[9] N. Boumal, B. Mishra, P.-A. Absil, and R. Sepulchre, "Manopt, a Matlab toolbox for optimization on manifolds," *Journal of Machine Learning Research*, vol. 15, no. 42, pp. 1455–1459, 2014. [Online]. Available: https://www.manopt.org

APPENDIX

The convex and non-convex settings are implemented in `Python 3.9`, using `PyTorch 1.8.0` as core library. Experiments are performed on `iPython` notebooks.

The Riemannian setting is implemented in `MatLab 2020b` using the `manopt` toolbox [9].

All the code necessary to reproduce the results presented next is found in the Github repository, link here.