ACHOO: ACTUALLY HIGHER ORDER OPTIMIZATION

Clayton W. Thorrez
claytonthorrez@gmail.com

ABSTRACT

In the field of mathematical optimization, especially in optimization for machine learning, the term "higher order optimization" is sometimes used to describe techniques which employ information beyond the first derivative of the objective function. However, in many of these cases, the so called "higher order" simply means 2nd or 3rd. In the unbounded space of integers, it seems fairly underwhelming to consider these "high". In this work we explore using **Ac**tually **H**igher **O**rder **O**ptimization (AcHOO) methods to solve common optimization problems in machine learning. We derive these methods from the basis of Taylor expansions, demonstrate optimization up to the 11th order on synthetic and real world datasets, and provide an implementation in python.

1 MOTIVATION

There have been many recent advances in machine learning. [1] The models are bigger, the data is bigger, the VC funding is bigger, but there is one area in ML which has lagged behind the other in terms of scaling. The vast majority of deep learning models today are trained using first order optimizers.

A few brave souls venture into the realm of using second or third order information and this is considered “higher order” methods by the community. Among the works accepted to the Neurips 2019 Higher Order Methods workshop and the Order UP 2022 Conference (HOO-2022), the large majority cover only second order optimization and only a handful dare to try third or fourth order. This work pushes the conventional boundaries of optimization order and introduces new State Of The Art (SOTA) optimization orders.

2 BACKGROUND

2.1 ANCIENT HISTORY

The optimizers currently used today in machine learning tasks have roots in very old methods. First order optimization, most famously represented by gradient descent, was introduced close to 200 years ago. (Cauchy et al., 1847) More than 300 years ago Newton introduced his namesake method in the context of root-finding. (Newton, 1711) When using this method to find the 0’s of a gradient, then the method becomes a second order optimizer. (Ypma, 1995) Developed after Newton’s method though published earlier, Edmund Halley used second derivative information in his own namesake method of root finding. (Halley, 1694) Similar to Newton’s method, when finding the roots of a gradient, then it moves up an order and becomes a third order method.

2.2 RECENT HISTORY

New optimization methods are an area of active research to this day, unfortunately it just seems to be going in the wrong direction. Since the 1960’s there has been research in "derivative free optimization", "evolutionary algorithms", and "black box optimization" all of which we group together as zero order optimization. (Nelder & Mead, 1965; Hansen & Ostermeier, 2001; Conn et al., 2009)

[1] I included this sentence since it seems to be the unofficial secret password to start an ML paper in current year.

These methods attempt to find the values which minimize a function, using only evaluations of the function itself, never taking it's derivative/gradient.

There are also now a wide range of first order variants in use including momentum (Nesterov, 1983), Adam, Adamax, (Kingma & Ba, 2014), AdamW (Loshchilov & Hutter, 2017), RAdam (Liu et al., 2020), and NAdam (Dozat, 2016) and even some non Adam variants if you can believe that.

Second order optimization has also enjoyed some further development, most notably adapting Newton's method into quasi-Netwon methods to scale better. The most famous of these being LBFGS. (Liu & Nocedal, 1989)

3 DERIVATIONS

There are several ways to understand and derive optimization methods. Here we will use a method involving Taylor approximations following Boyd et al. (2004) and we will see how it admits a relatively straightforward extension to *actually* higher orders.

3.1 TAYLOR SERIES

The standard way to express a Taylor series is using this equation.

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + ...$$

Where f is a function, x and a are points, $'$ indicates a derivative. (multiple $'$ indicate multiple levels of derivatives.) To make a k'th order approximation of a function, we can simply use the first k terms of the Taylor expansion. In the context of optimization it is more useful to rearrange the terms and extend to higher dimensions and this is the result.

$$f(x+v) = f(x) + v^T \nabla f(x) + v^T \frac{\nabla^2 f(x)}{2!} v + v^T \frac{\nabla^3 f(x)}{3!} vv + ...$$

Here we can interpret v as the update applied to the parameter vector x and it replaces $x - a$ in the previous equation. In the first it approximates the evaluation of a function at point x in terms of evaluations of the gradients at nearby points to x. Here it evaluates the function at an offset from a point x in terms of the gradient at the point x. The dimensions get a little weird in higher orders just know that there are as many v's multiplied together as the order.

3.2 FIRST ORDER

Ok now let's use this to derive gradient descent using a first order Taylor expansion.

$$f(x+v) \approx f(x) + v^T \nabla f(x)$$

In optimization we want to minimize f as much as possible. Which means we want to find v which makes $f(x+v)$ smaller. With a constant x, this is the same as making $v \nabla f(x)$ as negative as possible. According the Boyd et al. (2004) "A natural choice for the search direction is the negative gradient" thus $v = -\nabla f(x)$ and the update becomes:

$$x_{t+1} = x - \eta \nabla f(x)$$

Where η is the step size or learning rate introduced to prevent it from taking too large of a step into regions of parameter space where the first order Taylor approximation does not hold well.

3.3 SECOND ORDER

Ok now the same thing with second order Taylor expansion.

$$f(x+v) \approx f(x) + v^T \nabla f(x) + v^T \frac{\nabla^2 f(x)}{2!} v$$

Again the goal is to pick v which minimizes $f(x+v)$. We do this by finding a v such that $x+v$ is a stationary point so we differentiate the right hand side with respect to v and set to 0.

$$\frac{\partial}{\partial v}\left(f(x) + v^T \nabla f(x) + v^T \frac{\nabla^2 f(x)}{2!} v\right) = \nabla f(x) + v^T \nabla^2 f(x) = 0$$

Then solve for v and get

$$v = -\nabla^2 f(x) \nabla f(x)$$

Where $\nabla^2 f(x)$ is a square $D \times D$ matrix also called the Hessian (H). So in Newton's method, we change out the scalar η from gradient descent and replace it with the Hessian to provide an update with much better information about the local curvature of the objective surface.

Thus the update equation is:

$$x_{t+1} = x - H^{-1} \nabla f(x)$$

A weakness of Newton's method is that it requires taking the inverse of the Hession which can be a very expensive operation if the dimensionality D is large.

3.4 Third Order

This section draws heavily from a very nice stackoverflow answer explaining Halley's Method wyer33 (https://math.stackexchange.com/users/33022/wyer33). The basic idea is to do Newton's method on the gradient of the second order Taylor expansion, and make an approximation by substituting in the Newton update direction to make it solvable. Here $F(x)$ is the gradient, ie $\nabla f(x)$.

$$F(x+v) \approx F(x) + v^T \nabla F(x) + v^T \frac{\nabla^2 F(x)}{2} v$$

Then replace one of the v's with the Newton direction denoted v_2. (And v_1 will denote the gradient going forward).

$$F(x+v) \approx F(x) + v^T \nabla F(x) + v^T \frac{\nabla^2 F(x)}{2} v_2$$

Setting the right side to 0 (because the whole thing is a gradient) to get:

$$\left(\nabla F(x) + \frac{\nabla^2 F(x)}{2} v_2\right) v = -F(x)$$

Then back substitute $\nabla f(x)$ for $F(x)$ to get:

$$\left(\nabla^2 f(x) + \frac{\nabla^3 f(x)}{2} v_2\right) v = -\nabla f(x)$$

Finally solve for v

$$v = -\left(\nabla^2 f(x) + \frac{\nabla^3 f(x)}{2} v_2\right)^{-1} \nabla f(x)$$

The part inside the parenthesis which is inverted provides a clue as to how we will extend past order 3. We can see the first term is simply the Hessian, a $D \times D$ matrix. The second term is comprised of three parts. The inverse factorial coefficient of $\frac{1}{2}$, the direction vector of the lower order, in this case the second order v_2, and the $D \times D \times D$ tensor which we call the "thressian". (third order Hessian). It turns out that multiplying the $D \times D \times D$ tensor with the D dimensional vector yields a $D \times D$ result meaning it can add naturally with the Hessian term and multiply naturally with the gradient on the outside of the parenthesis.

3.5 To Infinity And Beyond

Generalizing on the pattern discovered in the third order solution, we can form a recursive update equation for any arbitrary order which utilizes high dimensional curvature of the objective landscape to formulate very precisely optimal updates.

Let $v_1 = -\nabla f(x)$. Then $v_2 = -\left(\nabla^2 f(x)\right)^{-1} \nabla f(x) = H^{-1} v_1$ which is the Newton update direction. We also use the above definition to see the third order update.

$$v_3 = \left(\nabla^2 f(x) + \frac{1}{2}\nabla^3 v_2\right)^{-1} v_1$$

Taking it another step and we get:

$$v_4 = \left(\nabla^2 f(x) + \frac{1}{2}\nabla^3 v_2 + \frac{1}{6}\nabla^4 v_3 v_2\right)^{-1} v_1$$

As the order get's higher and higher, the tensor of partial derivatives keeps getting more dimensions. Thus we repeatedly multiply by lower order update vectors to reduce the dimension to $D \times D$.

One more to show the point.

$$v_5 = \left(\nabla^2 f(x) + \frac{1}{2}\nabla^3 v_2 + \frac{1}{6}\nabla^4 v_3 v_2 + \frac{1}{24}\nabla^5 v_4 v_3 v_2\right)^{-1} v_1$$

Thus the general format is to compute the gradient tensor, multiply by the factorial coefficient, multiply by previous direction vectors down to v_2, sum up the intermediate $D \times D$ terms, invert, and multiply by the gradient. Boom goes the dynamite.

3.6 Disclaimer

We might have made some mistakes in either sign, transpose notation or other minor detail but the core idea is actually legit.

4 Experiments

4.1 Convergence on Synthetic Data

In order to test AcHOO, we implemented the general procedure described above using the jax python library which has super nice support for higher order derivatives by simply calling `jacfwd()` and `jacrev()` repeatedly. Our code is available at https://github.com/cthorrez/AcHOO.

For the first experiment we test on a synthetic linear regression dataset. We generate 10 rows with 1 feature per row. The features are from a standard Gaussian, then we also sample a weight coefficient from a standard Gaussian, and generate the labels by multiplying the features by the weights and adding Gaussian noise with standard deviation 0.2. We can optionally concatenate a column of ones to the input to allow for fitting an intercept term as well. Since Newton's method solves standard linear regression with mean squared error in closed form in a single iteration, we used a modified loss function where the error was raised to the power of 8 rather than 2 to accentuate the differences in the higher order methods.

order	1	2	3	4	5	7	10	11
#iter	36	4	**3**	**3**	**3**	**3**	**3**	**3**
time (s)	**0.27**	0.28	0.48	0.9	1.68	9.70	258.11	795.70

Table 1: Number of iterations until convergence and wall clock time for different optimization orders on the synthetic the 8th order linear regression dataset.

In this case there was no improvement in number of iterations until convergence, but we can see clearly the drastic increase in runtime as order increases.

4.2 Real World Applications

In order to test AcHOO in a real world production level environment. We decided to test it on the Iris flower classification dataset (Fisher, 1936) accessed from the UCI machine learning dataset repository. (Dua & Graff, 2017) This dataset contains 150 rows with 3 features per row with three class options. We simplify the dataset by doing binary classification of class 0 vs class 1 or 2 which cuts the parameter number by one third.

order	1	2	3	4	5	7	10
#iter	27	3	3	**2**	**2**	**2**	**2**
time (s)	**0.47**	0.55	0.88	1.57	2.95	14.43	1825

Table 2: Number of iterations until convergence and wall clock time for different optimization orders on the binarized Iris dataset.

While the time scaling is certainly poor, we do actually see convergence in fewer iteration than Newton's and Halley's methods when using 4th order or higher optimization methods.

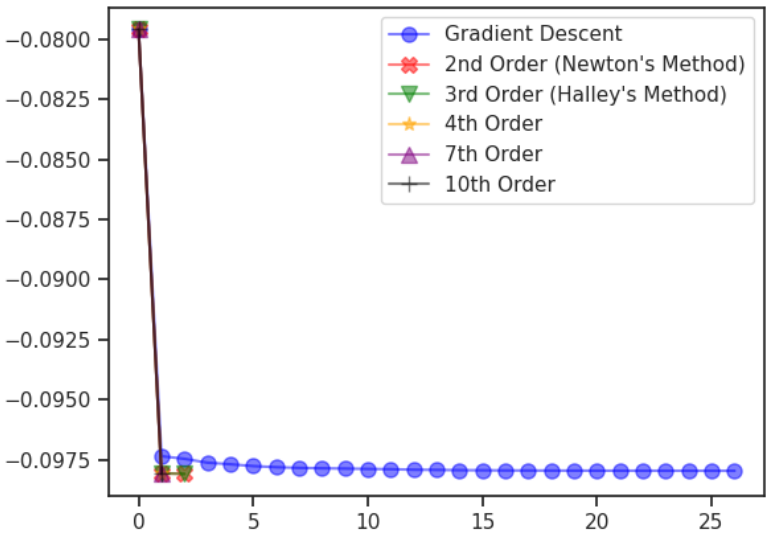

Figure 1: This doesn't really show much but I wrote the code for plotting the training loss so I'm putting it in.

5 DISCUSSION

5.1 STRENGTHS

Our number is bigger and thus SOTA.

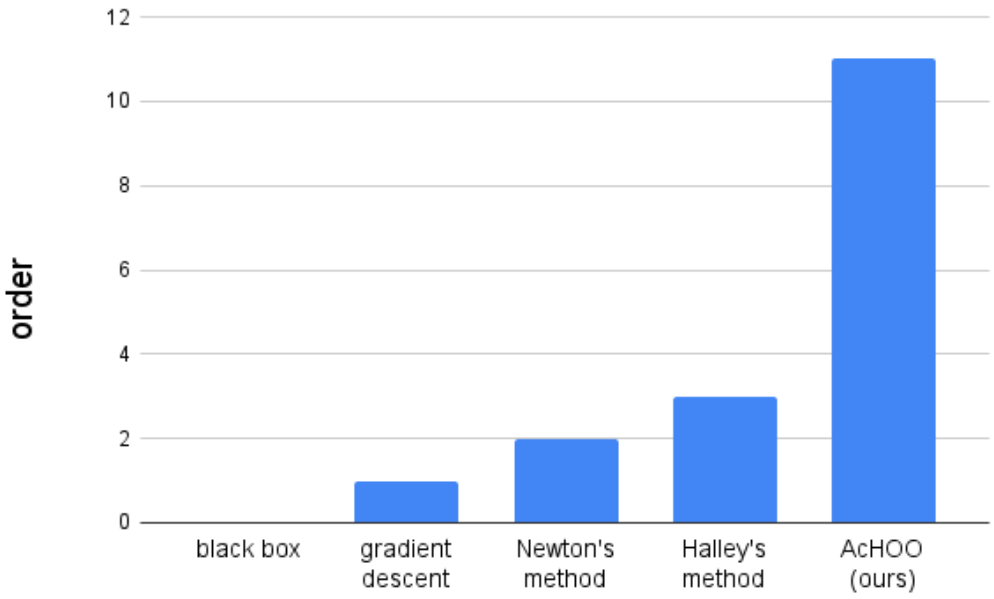

Figure 2: The order of different optimization methods. Eleven is just the highest presented in the current work, there is no theoretical upper bound on the framework introduced.

5.2 WEAKNESSES

Using higher order optimization methods has very high memory and time costs. For optimization of order k, the tensor of partial derivatives takes $\mathcal{O}(D^k)$ memory. For today's large models such as GPT3 (Brown et al., 2020) with 175 billion parameters, it would only take up to order 8 to surpass the estimated number of atoms in the universe. Even on our small scale experiments going beyond order 11 caused either out of memory errors or had prohibitively long runtimes.

5.3 NEXT STEPS

In the future we would like to take this same idea in the other direction. We briefly touched on zero order optimization using no gradient information. We would like to explore negative order optimization methods where we use point evaluations of the integral of a function to try to minimize the function.

5.4 CONCLUSION

In this work we introduced AcHOO, a framework to perform actually higher order optimization for machine learning. We included the methematical derivation for how to extend current optimization mehtods and proved the (in)effectiveness on synthetic and real world datasets.

REFERENCES

Stephen Boyd, Stephen P Boyd, and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *CoRR*, abs/2005.14165, 2020. URL https://arxiv.org/abs/2005.14165.

Augustin Cauchy et al. Méthode générale pour la résolution des systemes d'équations simultanées. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847.

Andrew R Conn, Katya Scheinberg, and Luis N Vicente. *Introduction to derivative-free optimization*. SIAM, 2009.

Timothy Dozat. Incorporating nesterov momentum into adam. 2016.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Ronald A Fisher. The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7 (2):179–188, 1936.

Edmund Halley. A new, exact, and easy method of finding the roots of any equations generally, and that without any previous reduction. *Philos. Trans. Roy. Soc. London*, 18:136–148, 1694.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgz2aEKDr.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2017.

John A Nelder and Roger Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

Yurii Evgen'evich Nesterov. A method of solving a convex programming problem with convergence rate o\bigl(k^2\bigr). In *Doklady Akademii Nauk*, volume 269, pp. 543–547. Russian Academy of Sciences, 1983.

Isaac Newton. *De analysi per aequationes numero terminorum infinitas*. 1711.

wyer33 (https://math.stackexchange.com/users/33022/wyer33). does it make sense to talk about third-order (or higher order) optimization methods? Mathematics Stack Exchange. URL https://math.stackexchange.com/q/2852107. URL:https://math.stackexchange.com/q/2852107 (version: 2018-07-15).

Tjalling J. Ypma. Historical development of the newton-raphson method. *SIAM Review*, 37(4): 531–551, 1995. ISSN 00361445. URL http://www.jstor.org/stable/2132904.