Neural Network Optimization using Stochastic Gradient Descent with Armijo Line Search

Alex Chaloner

December 2019

Abstract

In this report we review Stochastic Gradient Descent with Armijo Line Search as an optimization technique in Neural Networks. In particular, we inspect the work of Vaswani et al. [2019] and run similar experiments. We confirm Vaswani et al's findings that SGD with Armijo Line Search is a competitive optimizer that converges quickly and generalizes well. We critically inspect this work's implementations of SGD Line Search with momentum, find issues, and provide working alternatives. Our implementations of SGD Line Search with momentum are competitive.

1 Introduction

Stochastic Gradient Descent (SGD) is a popular method of optimizing Artificial Neural Networks (Goodfellow et al. [2016]). Success of SGD depends heavily on the setting of the learning rate (Bengio [2012]); if the learning rate is set too large then SGD does not converge, and if set too small SGD will take too long to converge. Random Search can be used to find a good learning rate among other network hyperparameters (Bergstra and Bengio [2012]).

Ruder [2016] identifies many popular SGD methods for Neural Networks, including Polyak Momentum (Polyak [1964]), Nesterov Accelerated Gradient, (Nesterov [1983], Nesterov [2004]), and adaptive gradient methods such as Adam (Kingma and Ba [2014]). More recently, there exists a competitive variant of Adam called Adabound (Luo et al. [2019]). We use Adam and Adabound as baselines to compare performance of our algorithms.

Line Search methods minimise convex functions (Nocedal and Wright [2006]). Given a starting point and a direction of movement, Line Search methods find a good step size for the direction of movement in a small number of iterations. Armijo line search, also known as Backtracking line search, is one such method. Line search can be used to select the learning rate at each step in Gradient Descent.

In this report we closely review the work of Vaswani et al. [2019] who use Line Search methods in combination with SGD methods. They claim their methods are robust to hyperparameter selection. We re-run a selection of their experiments and we find disagreements in computational timing. Furthermore, we find issue with their method of SGD and line search with momentum and we provide working alternatives.

2 Gradient Descent

Gradient Descent is a common optimization algorithm. The gradient of a function is used as a descent direction with the objective of finding a minima, although gradient descent can end anywhere with zero gradient including saddle points and maxima. It has been used to optimize Deep Neural Networks (Goodfellow et al. [2016]).

Du et al. [2019] show that Gradient Descent converges to zero training loss for overparameterized networks of Dense, Residual and Convolutional Residual Networks, using widths polynomial in the depth and number of data points. Practically, this is an intractable size for most data sets, but the result shows the theoretical effectiveness of Gradient Descent.

2.1 Stochastic Gradient Descent

Stochastic Gradient Descent is a variant of Gradient Descent. Stochastic Gradient Descent takes a random data point at each step (or a small batch of data points, typically 32, 64 or 128) (Ng [2019]) and calculates the loss function with respect to the chosen batch. The advantage of this is apparent for datasets containing a large number of data points (eg. CIFAR10 (Krizhevsky [2012]) contains 60000 data points); at every iterative step for batches of size b SGD needs only calculate the loss function and gradient of b data points, whereas full Gradient Descent calculates the loss and gradient on the entire dataset which can be magnitudes larger than b. The trade-off is that the stochastic gradient is not necessarily representative of the full gradient, meaning SGD can take steps that do not follow the direction of the minimal point of the full-dataset loss function, causing a larger number of iterations to find the minimum. Practically, the increase in computational speeds far outweighs the increase in iterations needed to converge (Ng [2019]), and many methods and heuristics have been used to improve SGD's effectiveness.

2.2 SGD with Momentum

SGD can be augmented with *momentum*. Momentum is known to be practically very effective, having basis in both intuition and theory (Goh [2017]). The two core momentum methods are Polyak momentum and Nesterov momentum. Polyak and Nesterov momentum use a momentum weight $0 < \gamma < 1$ to store an exponentially decaying weighted average of descent directions. The basic forms of these algorithms are:

Algorithm 1: Polyak Momentum

$v_0 \leftarrow 0$;
$w_0 \leftarrow arbitrary$;
for *k = 1, ..., T* **do**
  $v_k \leftarrow \gamma v_{k-1} + \eta \nabla f(w_{k-1})$;
  $w_k \leftarrow w_{k-1} - v_k$;
end

Algorithm 2: Nesterov Accelerated Gradient

$v_0 \leftarrow 0$;
$w_0 \leftarrow arbitrary$;
for *k = 1, ..., T* **do**
  $w'_{k-1} \leftarrow w_{k-1} - \gamma v_{k-1}$;
  $v_k \leftarrow \gamma v_{k-1} + \eta \nabla f(w'_{k-1})$;
  $w_k \leftarrow w_{k-1} - v_k$;
end

2.3 Adaptive Gradient Methods

Adaptive Gradient Methods are a family of optimization techniques that scale gradient updates with respect to the inverse magnitude of previous gradients. Intuitively, this balances the learning rate of each parameter so that they each descend at similar rates (Duchi et al. [2011]).

Adam (Kingma and Ba [2014]) is an Adaptive Gradient Method building on the strengths of RMSProp with momentum (Hinton and Tieleman [2012]). Adam records an exponentially weighted moving average of the first and second moments of the gradients, and corrects for their bias. In practise, Adam is fairly robust to hyperparameter choice (Goodfellow et al. [2016]).

AdaBound (Luo et al. [2019]) is a recent evolution of Adam, inspired by *gradient clipping*. AdaBound uses the same updates as Adam, except the learning rate is bounded element-wise by $[\eta_l, \eta_u]$. The paper specifies that over time η_l and η_u evolve and the bound becomes more constrained, so that the action of the AdaBound algorithm becomes more akin to SGD. This is shown to have theoretical convergence results as well as practical success.

3 Line Search

Given a function f, a parameter w and a direction p, line search methods aim to find a value of η so as to find a minimal value of $f(w + \eta p)$ (Nocedal and Wright [2006]). Practically, η is computed to

be an inexact solution; finding the exact solution is computationally expensive and often an inexact solution is good enough (Nocedal and Wright [2006]).

3.1 Armijo Line Search

Armijo Line Search (also known as Backtracking Line Search) aims to find a value of η satisfying the Armijo condition. The Armijo condition is as follows:

$$f(w_k + \eta p_k) \leq f(w_k) - c\eta \nabla f(w_k)^T p_k \tag{1}$$

Where $0 < c < 1$ is a constant.

The Armijo condition is usually paired with the *curvature condition* to form the Wolfe conditions. In Armijo Line Search, the curvature condition is omitted and a *backtracking* method is used in its place (Nocedal and Wright [2006]). The backtracking method 'backtracks' the step size in each iteration to ensure the step size doesn't get too small. Backtracking is computationally cheaper than the calculation of the curvature condition.

4 SGD with Armijo Line Search

4.1 SGD and Armijo

SGD+Armijo is a simple algorithm; on each iteration one finds a step size via Armijo line search, and uses this step size in an SGD update.

Vaswani et al. [2019] claim this to be practically very effective. They report the time cost per iteration is competitive with Adam and Adabound. Used as an optimizer for the datasets MNIST (Lecun et al. [1998]), CIFAR10, and CIFAR100 (Krizhevsky [2012]) they reach a significantly higher validation accuracy in less iterations.

Vaswani et al. [2019] prove convergence rates for the SGD+Armijo algorithm on functions f satisfying certain conditions. They state that f must be differentiable, have finite-sum structure, have a lower bound, be L-smooth, and be convex *or* non-convex and satisfying the strong growth condition. We focus on the practical results.

4.2 SGD and Armijo with Momentum

Vaswani et al. [2019] attempt SGD+Armijo with Polyak and Nesterov momentum. The authors perform Armijo line search on the plain SGD update followed by a Polyak (respectively Nesterov) momentum update. We believe this approach is wrong - the line search should be searching for the optimal step size for the momentum update rather than the plain SGD update. They report their Nesterov+Armijo optimizer diverges, although their Polyak+Armijo optimizer performs well. We believe their Nesterov+Armijo optimizer fails due to their method being wrong - indeed, our Nesterov+Armijo optimizer does succeed in converging. Our Polyak+Armijo optimizer is more computationally expensive than theirs but performs competitively.

Our proposed method is as follows. Let γ be the momentum value, and let v_k be the Polyak momentum value at the kth iteration. Define $w'_k = w_k - \gamma v_k$. Then the Polyak Armijo condition is:

$$f(w'_k - \eta \nabla f(w_k)) \leq f(w'_k) - c\eta (\nabla f(w'_k))^T \nabla f(w_k) \tag{2}$$

Similarly, letting v_k be the Nesterov momentum value at the kth iteration and analagously defining w'_k and γ, the Armijo condition for Nesterov updates is:

$$f(w'_k - \eta \nabla f(w'_k)) \leq f(w'_k) - c\eta \|\nabla f(w'_k)\|^2 \tag{3}$$

We note that the Nesterov condition (3) needs only calculate one gradient whereas the Polyak condition (2) requires two different gradients. This leads to extra time taken by the Polyak+Armijo algorithm, presented as follows:

Algorithm 3: Polyak+Armijo

Input : Number of batches B, Momentum γ, Initial weights w_0, Initial learning rate η_{init}, learning rate recovery λ, Armijo parameter c and Armijo decay β

$v_0 = 0$;
for *k = 0, ..., B-1* **do**
  $w'_k \leftarrow w_k - \gamma v_k$;
  $\eta \leftarrow reset(\eta, B, \lambda)$;
  `// Calculate` η `satisfying the Armijo condition`
  while $f_k(w'_k - \eta \nabla f_k(w_k)) > f_k(w'_k) - c\eta(\nabla f_k(w'_k))^T \nabla f_k(w_k)$ **do**
    $\eta \leftarrow \beta\eta$;
  end
  `// Polyak momentum updates`
  $v_{k+1} \leftarrow \gamma v_k + \eta \nabla f(w_k)$;
  $w_{k+1} \leftarrow w_k - v_{k+1}$;
end

$reset(\eta, B, \lambda)$ is some function to reset η on each iteration. Vaswani et al. [2019] create a heuristic reset method which slightly raises the value of eta each iteration which has the effect of quick line searches and allows for the learning rate to rise slowly over time, so as not to always decrease. This heuristic reset is

$$\eta \leftarrow \eta \lambda^{1/B}$$

where $\lambda > 1$ is a hyperparameter and B is the number of batches per epoch. We find this heuristic reset to be practically very effective and allows the SGD+Armijo algorithms to compete in computational costs with other optimization algorithms.

An analogous procedure exists for the Nesterov+Armijo algorithm using Nesterov updates and the Nesterov condition (3).

5 Experiments

5.1 Experimental Setup

We compare five optimization algorithms: SGD+Armijo, Polyak+Armijo, Nesterov+Armijo, Adam, and AdaBound. We adapt the code of Vaswani et al. [2019] to implement our corrected versions Polyak+Armiji and Nesterov+Armijo. We use two datsets; MNIST (Lecun et al. [1998]) and CIFAR10 (Krizhevsky [2012]).

For MNIST, we follow Vaswani et al. [2019] in using a Multilayer Perception (MLP) with one hidden layer of size 1000.

For CIFAR10, we follow Vaswani et al. [2019] in using ResNet34 (He et al. [2015]).

Each algorithm is trained for 50 epochs on each setup to compare the learning of each optimizer.

See Table 1 for hyperparameters used.

5.2 Experimental Results

The MNIST graphs show a stark contrast in the performance of Polyak+Armijo in comparison with other optimizers, both in accuracy and loss. It must be noted that Polyak+Armijo takes 1.27x the time of SGD+Armijo and 1.4x the time as Adabound; nonetheless, Polyak+Armijo is converges to 98.6% accuracy at epoch 25, which the other algorithms do not reach in twice the number of epochs - hence, time-wise, Polyak+Armijo is outperfoming the other optimizers.

The CIFAR10 graphs are more difficult to reach conclusions from, as the optimizers have not converged within 50 epochs. Nonetheless, between from epochs 30 to 50 we SGD+Armijo outperforming

Hyperparameter	SGD+Armijo	Polyak+Armijo	Nesterov+Armijo
Initial η	1.0	1.0	1.0
c	0.1	0.1	0.1
Momentum γ	-	0.6	0.6
Reset parameter λ	2	2	2
Batch size	128	128	128

Hyperparameter	Adam	Adabound
Initial η	0.01	0.01
β_1	0.9	0.9
β_2	0.999	0.999
Batch size	128	128

Table 1: Optimizer hyperparameters used

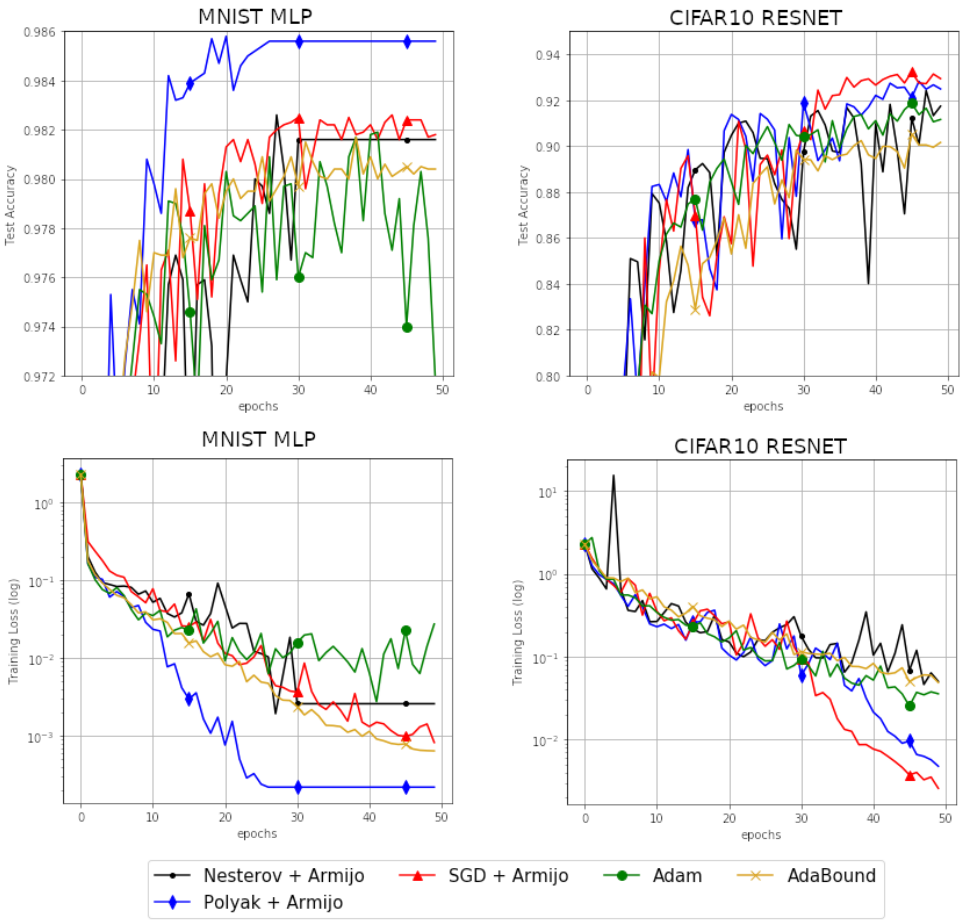

Figure 1: Test accuracy (top) and training loss (bottom) of optimizers on MNIST MLP and CIFAR10 ResNet34.

Optimizer	Average time per epoch (s)	
	MNIST MLP	CIFAR10 ResNet34
Adam	10.18	72.13
Adabound	10.55	80.09
SGD+Armijo	11.68	110.04
Polyak+Armijo	14.80	192.59
Nesterov+Armijo	13.86	150.00

Table 2: Optimizer computation times

the other optimizers closely followed by Polyak+Armijo. Again, computational costs of these methods are higher; SGD+Armijo takes 1.53x the time as Adam, and the other methods take over 2x the time. SGD+Armijo at epoch 30 is outperforming Adam and Adabound at epoch 50, and hence performs better time-wise.

Notably, Nesterov+Armijo does not stand out on either dataset, especially when controlled for its high computational costs. This is in contrast to how the Nesterov Accelerated Gradient technique outperforms Plain SGD. Nonetheless, we are happy that our Nesterov+Armijo converges, unlike the Nesterov+Armijo implemented by Vaswani et al. [2019].

6 Conclusions

In this report we have evaluated and added to the work of Vaswani et al. [2019]. We confirm that their SGD+Armijo method offers competition to other optimizers such as Adam and Adabound on the MNIST and CIFAR10 datasets. We critically examine their implementation of the Polyak+Armijo and Nesterov+Armijo optimizers and take issue with it; we derived new versions of these and empirically confirmed that they work. We find that Polyak+Armijo widely outperforms other optimizers on the MNIST dataset but takes disappointingly long on CIFAR10. We find that the Nesterov+Armijo optimizer works but offers little competitive value. We note that a different selection of hyperparameters might perform better.

References

Y. Bengio. Practical recommendations for gradient-based training of deep architectures, 2012.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, Feb. 2012. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=2188385.2188395`.

S. Du, J. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1675–1685, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL `http://proceedings.mlr.press/v97/du19c.html`.

J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 07 2011.

G. Goh. Why momentum really works. *Distill*, 2017. doi: 10.23915/distill.00006. URL `http://distill.pub/2017/momentum`.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

G. Hinton and T. Tieleman. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014.

A. Krizhevsky. Learning multiple layers of features from tiny images. *University of Toronto*, 05 2012.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86:2278 – 2324, 12 1998. doi: 10.1109/5.726791.

L. Luo, Y. Xiong, Y. Liu, and X. Sun. Adaptive gradient methods with dynamic bound of learning rate, 2019.

Y. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. *Dokl. Akad. Nauk SSSR*, 269:543–547, 1983.

Y. Nesterov. *Introductory Lectures on Convex Optimization*. Springer US, 2004.

A. Ng. Cs231n: Convolutional neural networks for visual recognition, optimization: Stochastic gradient descent, 2019. URL `https://cs231n.github.io/optimization-1/`.

J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Science+Business Media, 2 edition, 2006.

B. T. Polyak. Some methods of speeding up the convergence of iteration methods. 1964.

S. Ruder. An overview of gradient descent optimization algorithms, 2016.

S. Vaswani, A. Mishkin, I. Laradji, M. Schmidt, G. Gidel, and S. Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 3727–3740. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8630-painless-stochastic-gradient-interpolation-line-search-and-convergence-rates.pdf`.