On Stochastic Optimization Techniques

Billy Jun BILLYJUN@STANFORD.EDU
Lita Yang YANGLITA@STANFORD.EDU
Qian Liu QIANLLIU@STANFORD.EDU

Abstract

In recent years, there has been a resurgence of interest in the use of stochastic gradient descent (SGD) methods for numerous reasons: simplicity of the method, theoretically appealing generalization bound independent of number of training data, online learning from streaming information and application to large-scale learning. In this project, we explored several popular and recent SGD methods on a deep learning topology, and built on this survey to motivate a new hyperparameter-free variant that improves upon its original algorithm.

1. Introduction

Although similar with batch gradient descent (GD) methods in form, stochastic gradient descent (SGD) has many interesting and nonintuitive theoretical limits (Bottou, 2010). For example, the time it takes to reach accuracy ρ is $O(1/\rho)$, which unlike batch techniques is independent of the number of training examples. Even more nonintuitive is that second order information only improves the constant but not the asymptotic behavior. Lastly, despite its worse optimization performance on the empirical error than the batch GD, SGD offers a stronger generalization bound (asymptotically more efficient), which makes the study of SGD very interesting.

Recently, SGD was used to very successfully train a deep neural network (DNN) from random weights with the addition of classical momentum (CM) and Nesterov's Accelerated Gradient (NAG) (Sutskever et al., 2013). In a slightly different approach, SGD was revamped by scheduling adaptive learning rates by preconditioning (Duchi et al., 2011) and minimizing expected loss (Schaul et al., 2013). The main advantage of these adaptive techniques is that they offer stronger convergence near an optimum, which first-order SGD lacks.

However, given the numerous promising variants, it is unclear which method to use in practice. To resolve this confusion, the first part of this project will train the same architecture across several selected SGD algorithms and present optimization properties such as the learning rate curves and error statistics as a survey.

Another problem with methods such as SGD's is that there are hyperparameters that depend on a particular dataset and/or an architecture. This means a user has to sweep through a set of values and manually decide the hyperparameter setting before training. Not only is this cumbersome, but also takes a lot of time. Variance-based SGD (Schaul et al., 2013) is a hyperparameter free method, but we found that it converges relatively slowly, compared to other SGD algorithms when properly tuned. In the second part of the work, we incorporate the recent NAG idea to this variance-based method to accelerate convergence. The new hyperparameter-free variant showed a noticeable improvement over the original algorithm.

2. Methods

2.1. Architecture

As a first step, we wanted to choose a deep learning topology that would be a good model to test the SGD algorithms on. It would be ideal to implement and compare various deep learning algorithms, but due to pressing time, we only implemented a convolutional neural network (CNN) with softmax activation to classify the MNIST handwritten digits dataset. Our architecture, illustrated in Figure 1, is adopted from similar work (Le et al., 2011) that compared the conjugate gradient, L-BFGS and first-order stochastic gradient descent methods.

Initially, we implemented everything on Matlab with vectorization. This deemed valuable in verifying the correctness of implementation. However, due to slow runtime, computation-heavy code was rewritten in CUDA. This resulted in a significant performance increase.

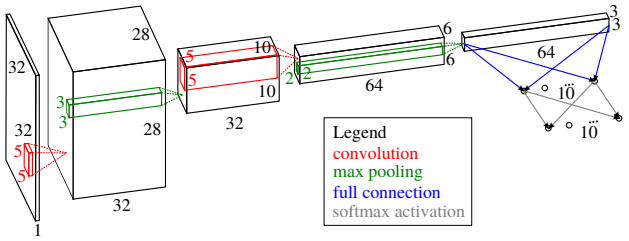

Figure 1. Illustration of the CNN architecture used in this work. We chose max pooling over uniform downsampling and mean pooling, because this captures the highest activity in the respective receptive field and backpropagation is more computationally efficient.

2.2. Stochastic gradient descent algorithms

We explored the following popular/recent algorithms from the SGD literature.

- Stochastic gradient descent (SGD1) with learning rate (Le et al., 2011): $\eta^{(t)} = \frac{\alpha}{\beta+t}$. This is popular because of its simple implementation.

- SGD with classical momentum (SGD-CM):

$$v^{(t+1)} = \mu v^{(t)} - \epsilon \nabla f(\theta^{(t)}) \tag{1}$$

$$\theta^{(t+1)} = \theta^{(t)} + v^{(t+1)}. \tag{2}$$

- SGD with classical momentum and Nesterov's Accelerated Gradient (SGD-CM-NAG) (Sutskever et al., 2013):

$$\mu^{(t)} = \min\{1 - 2^{-1-\log_2(1+\lfloor t/250 \rfloor)}, \mu_{\max}\} \tag{3}$$

$$v^{(t+1)} = \mu^{(t)} v^{(t)} - \epsilon \nabla f(\theta^{(t)} + \mu^{(t)} v^{(t)}) \tag{4}$$

$$\theta^{(t+1)} = \theta^{(t)} + v^{(t+1)}. \tag{5}$$

  The authors made a clear distinction between the two terms in (4), calling the first classical, as it turned out that NAG is another form of momentum that offers quadratic convergence rate for convex problems, which is unprecedented in other SGD methods. We used the same terminology in this work.

- Adaptive Subgradient SGD (ADAGRAD) (Duchi et al., 2011):

$$\eta_i^{(t)} = \epsilon \left(\mu + \sum_{s=0}^{t} \left(\nabla_{\theta_i^{(s)}} \right)^2 \right)^{-1/2} \tag{6}$$

  The original learning rate from the paper does not have the hyperparameter μ. This was added for other similar algorithms to prevent division-by-zero problem.

- Stochastic Diagonal Levenberg-Marquardt (SGD-LM) (LeCun):

$$\eta_i^{(t)} = \epsilon \left(\mu + \nabla^2_{\theta_i^{(t)}} \right)^{-1} \tag{7}$$

  This is a popular algorithm for training convolutional neural networks (CNN), so was included in this work.

- Local variance-based SGD (vSGD-l) (Schaul et al., 2013):

$$\eta_i^{(t)} = \frac{(\mathbb{E}[\nabla_{\theta_i}])^2}{\mu + \nabla^2_{\theta_i^{(t)}} \mathbb{E}[(\nabla_{\theta_i})^2]}, \tag{8}$$

  This algorithm is derived from the principle of minimum mean squared error (MMSE) that aims to greedily minimize per-sample quadratic loss. It is completely hyperparameter free (μ is just a small constant for preventing division by zero).

2.3. Training

The MNIST digits dataset was first preprocessed so that each feature is a Gaussian random variable with zero mean and unit variance. Then, the images were padded with 0 around the perimeter to have a new dimension of 32×32 pixels. Because we were interested in measuring the raw performance of SGD methods, techniques such as minibatching and pretraining were not performed. For each algorithm, we trained for 15 epochs on NVIDIA GeForce GTX 480 with random weight initialization ($\sim \mathcal{N}(0, 10^{-4})$). Then, twice per epoch, we checked its mean classification error on a 20% hold-out validation set (12000 images) and saved the weights when a better model is learned. This final model is then used to make predictions on the test set. Since our primary focus was on how well optimization techniques reduce empirical errors, early stopping and the natural sparsity of the CNN were the only form of regularization used to improve test error.

For simplicity, backpropagation of diagonal Hessian estimates in our CUDA implementation is performed after the gradient signals are backpropagated. However, in our report, we assumed that this can be performed in parallel, and thus we rescaled run-time to that of the earliest finishing algorithm.

3. Survey of popular/recent work

First, we verified that our CUDA implementation is correct by visualizing the kernels it learned (Figure 2). Then, to make certain all optimization techniques are performing at their best, we ran through a sweep

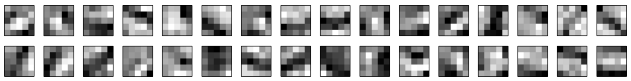

Figure 2. Verification of the CNN implementation was performed with SGD1 due to its simplicity and obvious correctness.

Algorithms	Parameters
SGD1	$\alpha = 0.01 \times \lvert\mathcal{D}\rvert$, $\beta = \lvert\mathcal{D}\rvert$
SGD-CM	$\epsilon = 10^{-4}, \mu = 0.99$
SGD-CM-NAG	$\epsilon = 10^{-4}, \mu_{\max} = 0.99$
ADAGRAD	$\epsilon = 10^{-2}, \mu = 10^{-6}$
SGD-LM	$\epsilon = 10^{-3}, \mu = 10^{-2}$
vSGD-l	$\mu = 10^{-6}$

Table 1. Results of hyperparameter selection.

of values to find a set of good hyperparameters and chose the settings summarized in Table 1.

As an interesting practical guide to hyperparamter selection, we found that ADAGRAD and vSGD-l are insensitive to the choice of epsilon value μ unlike SGD-LM (Figure 3). If the model is regularized, for example, with an L2 norm, then division-by-zero error is no longer an issue with SGD-LM. But, this would mean that the optimal hyperparameter ϵ becomes a function of the regularization hyperparameters as well.

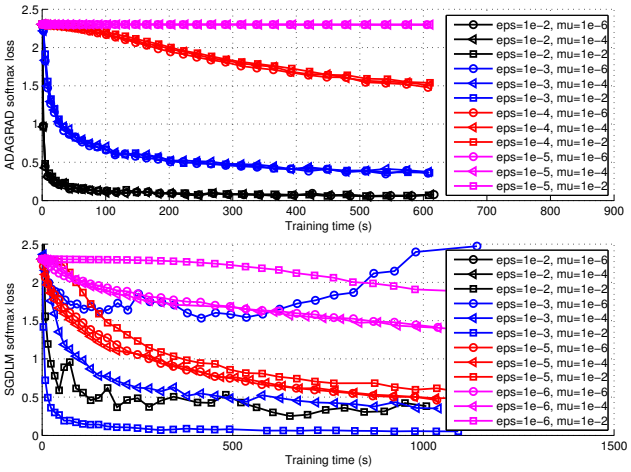

Figure 3. Increasing μ tends to help SGD-LM (bottom) become more numerically stable, although it can partially reduce the convergence rate.

After training the same architecture with sweep-selected hyperparameters, we obtained the learning rate curves in Figure 4, and their respective error statistics in Table 2. Apart from vSGD-l, which is of different nature, SGD1 as expected had the worst empirical error. This was mainly because of the predetermined learning schedule, which allows for fast initial convergence but later suffers from a slow-down near a local optimum. This also explains why SGD1 had the best test error (Table 2); it just overfit less. In this respect, the adaptive rates of ADAGRAD and SGD-LM led to better empirical errors. However, for purely optimizing a function, SGD methods with momentum had the best performance. In particular, NAG, which enjoys quadratic convergence rate for convex problems (Sutskever et al., 2013), converges increasingly better as it approaches the minimum, because the problem becomes more convex locally.

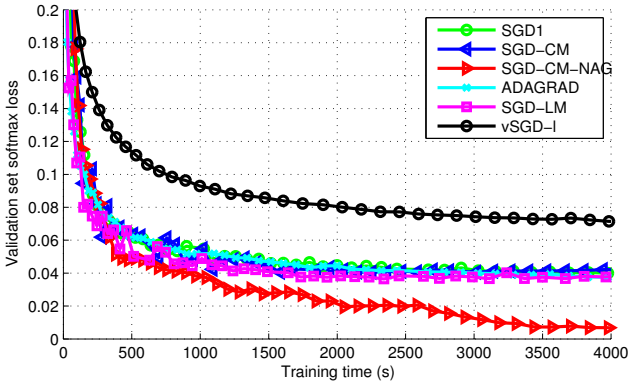

Figure 4. The actual computation time for vSGD-l and SGD-LM on CUDA took longer because of the backpropagation of diagonal Hessian estimates, but we assumed that this can be performed in parallel in this plot.

Algorithms	Validation set MCE	Test set MCE
SGD1	1.08%	**0.91**%
SGD-CM	0.84%	1.00%
SGD-NAG	**0.07**%	1.28%
ADAGRAD	0.91%	0.97%
SGD-LM	0.99%	0.93%
vSGD-l	1.91%	1.70%

Table 2. Early stopping and sparsity of the CNN are the only form of regularization used to generate these results.

In the original paper (Schaul et al., 2013), it was shown that SGD1 has a limited lower bound to which objective can be reduced, and that vSGD methods outperform on the MNIST dataset. However, this was because the decay rate of SGD1 was chosen to be $\lambda = 1$. With our choice of hyperparameters, α and β (Table 1), the learning rate schedule becomes

$$\eta^{(t)} = \frac{\alpha}{\beta + t} = \frac{0.01|\mathcal{D}|}{|\mathcal{D}| + t} = \frac{0.01}{1 + (1/|\mathcal{D}|)t} =: \frac{\eta_0}{1 + \lambda t}$$

The decay factor $\lambda = \frac{1}{|\mathcal{D}|} < 1$ allows the learning rate to decrease slowly over the number of epochs rather

than the total number of iterations. This allowed SGD1 to actually converge to a smaller objective value and result in lower mean classification error rates than vSGD-l.

Although vSGD-l has the advantage of being hyperparameter-free, it does not perform as well as the other methods when their hyperparameters are manually tuned to be optimal. We speculate that this is because vSGD-l chooses the optimal learning rate based on the minimum mean squared error (MMSE) like the Kalman filter, but the squared loss assumption does not hold strongly for this particular problem. Furthermore, upon probing various dependencies of the learning rate computation, it was found that the inverse of memory size, which ultimately decides how much of the current gradient information gets incorporated into the current estimates of $\mathbb{E}[\nabla_{\theta_i}]$ and $\mathbb{E}[\nabla^2_{\theta_i}]$, diminshes very rapidly (Figure 5).

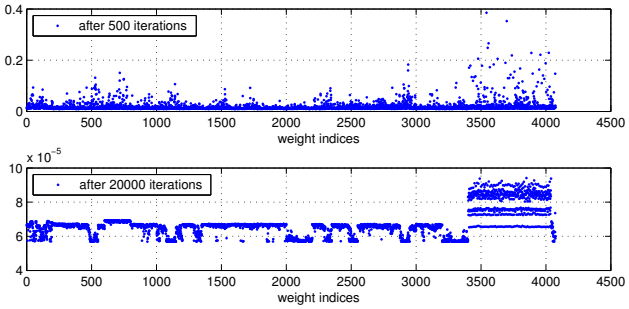

Figure 5. y-axis represents the ratio of the current gradient information used in estimating $\mathbb{E}[\nabla_{\theta_i}]$ and $\mathbb{E}[\nabla^2_{\theta_i}]$. About a half way through the first epoch, this fraction is practically 0.

4. Improving vSGD-l

In the previous section, we saw that vSGD-l converges relatively weakly, compared to other techniques with careful hyperparameter selection. However, vSGD-l is appealing because it is truly hyperparameter free. Observing that classical momentum and Nesterov's Accelerated Gradient offers quadratic convergence near an optimum (due to increased convexity), we applied this idea to accelerate vSGD-l.

As a first attempt, the original NAG formulation (Sutskever et al., 2013) with momentum schedule (3) was applied directly without any modifications. As shown in Figure 6, NAG in its unaltered form actually degrades vSGD-l. This is because vSGD-l implements the slow-start rule (Schaul et al., 2013) so that the algorithm better estimates gradient and hessian expectations initially. However, because of the hard-coded value of 250 in (3), $\mu^{(t)}$ quickly saturates to $\mu_{\max}$. Consequently, the initial expectation estimates lose their accuracies due to the distortion induced by a quickly growing momentum; learning rates end up being suboptimal.

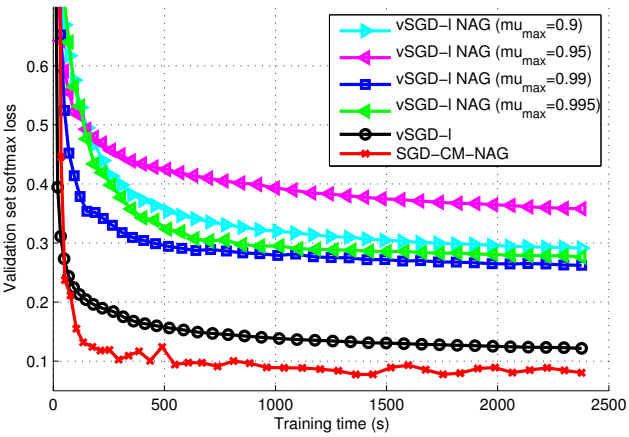

Figure 6. NAG destroys the initial local estimates of $\mathbb{E}[\nabla_{\theta_i}]$ and $\mathbb{E}[(\nabla_{\theta_i})^2]$, and results in poorer performance. For this section, our CUDA implementation resulted in NaN errors when running on the full architecture in Figure 1. All experiments are performed with 8 and 16 kernel maps in the first two convolutional layers.

To cope with this, a new momentum schedule is proposed in Algorithm 1. The two major changes are the replacement of 250 by $|\mathcal{D}|$, the size of training data, and the removal of $\mu_{\max}$. This choice of the denominator no longer hard-codes a particular value and makes certain that the rate of momentum accumulation is kept small in the first epoch, allowing vSGD-l to gather accurate statistics for good initial estimates of expectation values. Here, we are implicitly assuming that there are much more than 250 training examples. If there are a small number of training data, then hyperparameter selection doesn't take long; other, more viable learning techniques such as SGD-CM-NAG can be applied without too much overhead. Under this scheme, it takes about 500 epochs for any training sets to reach a μ value of 0.999. Practically, by then, the vSGD-l learning rates become negligible and the classical momentum term never diverges. However, we have yet to prove theoretically that this is so.

We tested this idea for 15 epochs on a reduced architecture of 8 and 16 kernel maps in the first two convolutional layers, because the CUDA implementation of vSGD-l NAG resulted in an NaN error on the full-sized CNN. In summary, the new hyperparameter-free algorithm reduced the objective value of vSGD-l by a factor of about 35% (Figure 7) and decreased the empirical and validation errors by an additional 1%.

Algorithm 1 vSGD-l NAG with a new momentum schedule; initialization is the same as vSGD-l.

repeat
  draw a sample $\{x^{(t)}, y^{(t)}\} \sim \mathcal{D}$
  $\mu \leftarrow 1 - 2^{-1-\log_2(1+\lfloor t/|\mathcal{D}| \rfloor)}$
  $g^{(t)} \leftarrow \nabla f(\theta + \mu v; x^{(t)}, y^{(t)})$
  $h^{(t)} \leftarrow \nabla^2 f(\theta + \mu v; x^{(t)}, y^{(t)})$
  for $i \in [1, \ldots, d]$ **do**
    $\bar{g}_i \leftarrow (1 - \tau_i^{-1})\bar{g}_i + \tau_i^{-1} g_i^{(t)}$
    $\bar{v}_i \leftarrow (1 - \tau_i^{-1})\bar{v}_i + \tau_i^{-1} \left(g_i^{(t)}\right)^2$
    $\bar{h}_i \leftarrow (1 - \tau_i^{-1})\bar{h}_i + \tau_i^{-1} |h_i^{(t)}|$
    $\tau_i \leftarrow \left(1 - \frac{(\bar{g}_i)^2}{\bar{v}_i}\right)\tau_i + 1$
    $v_i \leftarrow \mu v_i - \frac{(\bar{g}_i)^2}{\bar{h}_i \bar{v}_i} g_i^{(t)}$
    $\theta_i \leftarrow \theta_i + v_i$
  end for
until convergence requirement is met

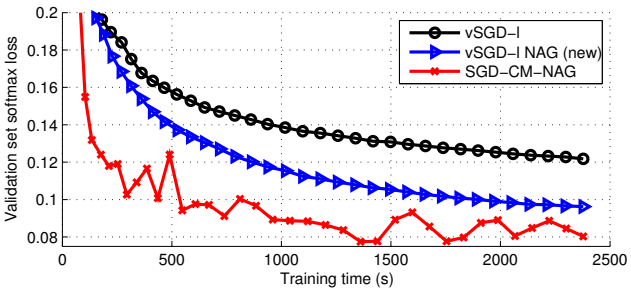

Figure 7. SGD-CM-NAG still outperforms both algorithms, but vSGD-l NAG closes the gap considerably.

Algorithms	Validation set MCE	Test set MCE
vSGD-l	3.71%	3.14%
vSGD-l NAG	2.78%	2.41%
SGD-CM-NAG	**2.27%**	**2.13%**

Table 3. Summary of error statistics

5. Summary

In this project, a CNN-softmax topology was trained by several popular and recent SGD variants to classify the MNIST handwritten digits dataset. From this survey, it was experimentally confirmed that NAG momentum minimizes the objective the best with its quadratic convergence rate property for convex problems; this effect shows up more dominantly, as it gets closer to a local optimum in which the problem seems more convex. The local-variance-based SGD (vSGD-l) is hyperparameter-free but it was found that it does not enjoy the same level of convergence and generalization as other methods with manual tuning. In the second part of this work, a new momentum schedule was proposed to combine the strengths of local-variance estimation and Nesterov's acceleration, which resulted in a new hyperparameter-free algorithm that outperformed the original vSGD-l. However, there still remain some practical difficulties in using vSGD-l NAG; 1. it is not trivial to implement, 2. it requires diagonal Hessian estimates, and 3. the memory overhead is huge. In particular, it needs to additionally store 5-6 times the number of parameters. In a deep learning architecture with millions of parameters, this may be impractical, or will require more hardware to run.

Acknowledgments

The authors would like to thank Andrew Maas, Brody Huval and Kyle Anderson for helpful discussions and advice on function optimization techniques to consider.

References

Bottou, Léon. Large-scale machine learning with stochastic gradient descent. In Lechevallier, Yves and Saporta, Gilbert (eds.), *Proceedings of the 19th International Conference on Computational Statistics (COMPSTAT'2010)*, pp. 177–187, Paris, France, August 2010. Springer.

Duchi, John, Hazan, Elad, and Singer, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12: 2121–2159, July 2011. ISSN 1532-4435.

Le, Quoc, Ngiam, Jiquan, Coates, Adam, Lahiri, Ahbik, Prochnow, Bobby, and Ng, Andrew. On optimization methods for deep learning. In Getoor, Lise and Scheffer, Tobias (eds.), *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, ICML '11, pp. 265–272, New York, NY, USA, June 2011. ACM. ISBN 978-1-4503-0619-5.

LeCun, Yann. More optimization methods. pp. 38. URL http://cilvr.cs.nyu.edu/diglib/lsml/lecture02-optimi.pdf.

Schaul, Tom, Zhang, Sixin, and LeCun, Yann. No More Pesky Learning Rates. In *International Conference on Machine Learning (ICML)*, 2013.

Sutskever, Ilya, Martens, James, Dahl, George, and Hinton, Geoffrey. On the importance of initialization and momentum in deep learning. In Dasgupta, Sanjoy and McAllester, David (eds.), *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pp. 1139–1147. JMLR Workshop and Conference Proceedings, May 2013.