Stochastic gradient descent

From Wikipedia, the free encyclopedia

Stochastic gradient descent (often shortened in **SGD**), also known as **incremental** gradient descent, is a stochastic approximation of the gradient descent optimization method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration.

Contents

Background

Both statistical estimation and machine learning consider the problem of minimizing an objective function that has the form of a sum:

$$Q(w) = \frac{1}{n}\sum_{i=1}^{n} Q_i(w),$$

where the parameter w which minimizes $Q(w)$ is to be estimated. Each summand function Q_i is typically associated with the i-th observation in the data set (used for training).

In classical statistics, sum-minimization problems arise in least squares and in maximum-likelihood estimation (for independent observations). The general class of estimators that arise as minimizers of sums are called M-estimators. However, in statistics, it has been long recognized that requiring even local minimization is too restrictive for some problems of maximum-likelihood estimation.[1] Therefore, contemporary statistical theorists often consider stationary points of the likelihood function (or zeros of its derivative, the score function, and other estimating equations).

The sum-minimization problem also arises for empirical risk minimization: In this case, $Q_i(w)$ is the value of the loss function at i-th example, and $Q(w)$ is the empirical risk.

When used to minimize the above function, a standard (or "batch") gradient descent method would perform the following iterations :

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q_i(w)/n,$$

where η is a step size (sometimes called the *learning rate* in machine learning).

In many cases, the summand functions have a simple form that enables inexpensive evaluations of the sum-function and the sum gradient. For example, in statistics, one-parameter exponential families allow economical function-evaluations and gradient-evaluations.

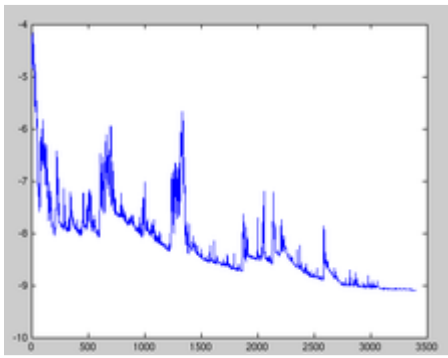
However, in other cases, evaluating the sum-gradient may require expensive evaluations of the gradients from all summand functions. When the training set is enormous and no simple formulas exist, evaluating the sums of gradients becomes very expensive, because evaluating the gradient requires evaluating all the summand functions' gradients. To economize on the computational cost at every iteration, stochastic gradient descent samples a subset of summand functions at every step. This is very effective in the case of large-scale machine learning problems.[2]

Iterative method

In stochastic (or "on-line") gradient descent, the true gradient of $Q(w)$ is approximated by a gradient at a single example:

$$w := w - \eta \nabla Q_i(w).$$

As the algorithm sweeps through the training set, it performs the above update for each training example. Several passes can be made over the training set until the algorithm converges. If this is done, the data can be shuffled for each pass to prevent cycles. Typical implementations may use an adaptive learning rate so that the algorithm converges.



Fluctuations in the total objective function as gradient steps with respect to mini-batches are taken.

In pseudocode, stochastic gradient descent can be presented as follows:

- Choose an initial vector of parameters w and learning rate η.
- Repeat until an approximate minimum is obtained:
  - Randomly shuffle examples in the training set.
  - For $i = 1, 2, \ldots, n$, do:
    - $w := w - \eta \nabla Q_i(w)$.

A compromise between computing the true gradient and the gradient at a single example, is to compute the gradient against more than one training example (called a "mini-batch") at each step. This can perform significantly better than true stochastic gradient descent because the code can make use of vectorization libraries rather than computing each step separately. It may also result in smoother convergence, as the gradient computed at each step uses more training examples.

The convergence of stochastic gradient descent has been analyzed using the theories of convex minimization and of stochastic approximation. Briefly, when the learning rates η decrease with an appropriate rate, and subject to relatively mild assumptions, stochastic gradient descent converges almost surely to a global minimum when the objective function is convex or pseudoconvex, and otherwise converges almost surely to a local minimum.[3][4] This is in fact a consequence of the Robbins-Siegmund theorem.[5]

Example

Let's suppose we want to fit a straight line $y = w_1 + w_2 x$ to a training set of two-dimensional points $(x_1, y_1), \ldots, (x_n, y_n)$ using least squares. The objective function to be minimized is:

$$Q(w) = \sum_{i=1}^{n} Q_i(w) = \sum_{i=1}^{n} (w_1 + w_2 x_i - y_i)^2.$$

The last line in the above pseudocode for this specific problem will become:

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} := \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} - \eta \begin{bmatrix} 2(w_1 + w_2 x_i - y_i) \\ 2x_i(w_1 + w_2 x_i - y_i) \end{bmatrix}.$$

Applications

Stochastic gradient descent is a popular algorithm for training a wide range of models in machine learning, including (linear) support vector machines, logistic regression (see, e.g., Vowpal Wabbit) and graphical models.[6] When combined with the backpropagation algorithm, it is the *de facto* standard algorithm for training artificial neural networks.[7]. Its use has been also reported in the Geophysics community, specifically to applications of Full Waveform Inversion (FWI) [8].

Stochastic gradient descent competes with the L-BFGS algorithm, which is also widely used. Stochastic gradient descent has been used since at least 1960 for training linear regression models, originally under the name ADALINE.[9]

Another popular stochastic gradient descent algorithm is the least mean squares (LMS) adaptive filter.

Extensions and variants

Many improvements on the basic stochastic gradient descent algorithm have been proposed and used. In particular, in machine learning, the need to set a learning rate (step size) has been recognized as problematic. Setting this parameter too high can cause the algorithm to diverge; setting it too low makes it slow to converge. A conceptually simple extension of stochastic gradient descent makes the learning rate a decreasing function η_t of the iteration number t, giving a *learning rate schedule*, so that the first iterations cause large changes in the parameters, while the later ones do only fine-tuning. Such schedules have been known since the work of MacQueen on k-means clustering.[10]

Momentum

Further proposals include the *momentum method*, which appeared in Rumelhart, Hinton and Williams' seminal paper on backpropagation learning.[11] Stochastic gradient descent with momentum remembers the update Δw at each iteration, and determines the next update as a convex combination of the gradient and the previous update:[12]

$$\Delta w := \eta \nabla Q_i(w) + \alpha \Delta w$$
$$w := w - \Delta w$$

or as a mathematically equivalent formulation:[13]

$$\Delta w := -\eta \nabla Q_i(w) + \alpha \Delta w$$
$$w := w + \Delta w$$

that leads to:

$$w := w - \eta \nabla Q_i(w) + \alpha \Delta w$$

where the parameter w which minimizes $Q(w)$ is to be estimated, and η is a step size (sometimes called the *learning rate* in machine learning).

The name momentum stems from an analogy to momentum in physics: the weight vector, thought of as a particle traveling through parameter space,[11] incurs acceleration from the gradient of the loss ("force"). Unlike in classical stochastic gradient descent, it tends to keep traveling in the same direction, preventing oscillations. Momentum has been used successfully for several decades.[14]

Averaging

Averaged stochastic gradient descent, invented independently by Ruppert and Polyak in the late 1980s, is ordinary stochastic gradient descent that records an average of its parameter vector over time. That is, the update is the same as for ordinary stochastic gradient descent, but the algorithm also keeps track of[15]

$$\bar{w} = \frac{1}{t} \sum_{i=0}^{t-1} w_i.$$

When optimization is done, this averaged parameter vector takes the place of w.

AdaGrad

AdaGrad (for adaptive gradient algorithm) is a modified stochastic gradient descent with per-parameter learning rate, first published in 2011.[16][17] Informally, this increases the learning rate for more sparse parameters and decreases the learning rate for less sparse ones. This strategy often improves convergence performance over standard stochastic gradient descent in settings where data is sparse and sparse parameters are more informative. Examples of such applications include natural language processing and image recognition.[16] It still has a base learning rate η, but this is multiplied with the elements of a vector $\{G_{j,j}\}$ which is the diagonal of the outer product matrix.

$$G = \sum_{\tau=1}^{t} g_\tau g_\tau^\mathsf{T}$$

where $g_\tau = \nabla Q_i(w)$, the gradient, at iteration τ. The diagonal is given by

$$G_{j,j} = \sum_{\tau=1}^{t} g_{\tau,j}^2.$$

This vector is updated after every iteration. The formula for an update is now

$$w := w - \eta \, \mathrm{diag}(G)^{-\frac{1}{2}} \circ g$$[a]

or, written as per-parameter updates,

$$w_j := w_j - \frac{\eta}{\sqrt{G_{j,j}}} g_j.$$

Each $\{G_{(i,i)}\}$ gives rise to a scaling factor for the learning rate that applies to a single parameter w_i. Since the denominator in this factor, $\sqrt{G_i} = \sqrt{\sum_{\tau=1}^{t} g_\tau^2}$ is the ℓ_2 norm of previous derivatives, extreme parameter updates get dampened, while parameters that get few or small updates receive higher learning rates.[14]

While designed for convex problems, AdaGrad has been successfully applied to non-convex optimization.[18]

RMSProp

RMSProp (for Root Mean Square Propagation) is also a method in which the learning rate is adapted for each of the parameters. The idea is to divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight.[19] So, first the running average is calculated in terms of means square,

$$v(w,t) := \gamma v(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2$$

where, γ is the forgetting factor.

And the parameters are updated as,

$$w := w - \frac{\eta}{\sqrt{v(w,t)}} \nabla Q_i(w)$$

RMSProp has shown excellent adaptation of learning rate in different applications. RMSProp can be seen as a generalization of Rprop and is capable to work with mini-batches as well opposed to only full-batches.[20]

Adam

'Adam'[21] (for Adaptive Moment Estimation) is an update to *RMSProp* optimizer. In this running average of both the gradients and their magnitudes are used. The three equations that define this optimizer are as follows,

$$m(w,t) = \gamma_1 m(w,t-1) + (1-\gamma_1)\nabla Q_i(w)$$
$$v(w,t) = \gamma_2 v(w,t-1) + (1-\gamma_2)(\nabla Q_i(w))^2$$

$$\hat{m}(w,t) = \frac{1}{1-\gamma_1^t} m(w,t)$$
$$\hat{v}(w,t) = \frac{1}{1-\gamma_2^t} v(w,t)$$

$$w(t+1) = w(t) - \frac{\eta}{\sqrt{\hat{v}(w,t)} + \epsilon} \hat{m}(w,t)$$

where, γ_1 and γ_2 are two forgetting factors of the algorithm, respectively for gradients and magnitude of gradients.

kSGD

Kalman-based Stochastic Gradient Descent (kSGD)[22] is an online and offline algorithm for learning parameters from statistical problems from quasi-likelihood models, which include linear models, non-linear models, generalized linear models, and neural networks with squared error loss as special cases. For online learning problems, kSGD is a special case of the Kalman Filter for linear regression problems, a special case of the Extended Kalman Filter for non-linear regression problems, and can be viewed as an incremental Gauss-

Newton method. The benefits of kSGD, in comparison to other methods, are (1) it is not sensitive to the condition number of the problem ,[b] (2) it has a robust choice of hyperparameters, and (3) it has a stopping condition. The drawbacks of kSGD is that the algorithm requires storing a dense covariance matrix between iterations, and requires a matrix-vector product at each iteration.

To describe the algorithm, suppose $Q_i(w)$, where $w \in \mathbb{R}^p$ is defined by an example $(Y_i, X_i) \in \mathbb{R} \times \mathbb{R}^d$ such that

$$\nabla_w Q_i(w) = \frac{Y_i - \mu(X_i, w)}{V(\mu(X_i, w))} \nabla_w \mu(X_i, w)$$

where $\mu(X_i, w)$ is mean function (i.e. the expected value of Y_i given X_i), and $V(\mu(X_i, w))$ is the variance function (i.e. the variance of Y_i given X_i). Then, the parameter update, $w(t+1)$, and covariance matrix update, $M(t+1)$ are given by the following

$$\begin{aligned}
p &= \nabla_w \mu(X_{t+1}, w(t)) \\
m &= \mu(X_{t+1}, w(t)) \\
v &= M(t)p \\
s &= min\{\gamma_1, max\{\gamma_2, V(m)\}\} + v^\mathsf{T} p \\
w(t+1) &= w(t) + \frac{Y_{t+1} - m}{s} v \\
M(t+1) &= M(t) - \frac{1}{s} v v^\mathsf{T}
\end{aligned}$$

where γ_1, γ_2 are hyperparameters. The $M(t)$ update can result in the covariance matrix becoming indefinite, which can be avoided at the cost of a matrix-matrix multiplication. $M(0)$ can be any positive definite symmetric matrix, but is typically taken to be the identity. As noted by Patel,[22] for all problems besides linear regression, restarts are required to ensure convergence of the algorithm, but no theoretical or implementation details were given. In a closely related, off-line, mini-batch method for non-linear regression analyzed by Bertsekas,[23] a forgetting factor was used in the covariance matrix update to prove convergence.

Notes

a. $\circ$ is the element-wise product.
b. For the linear regression problem, kSGD's objective function discrepancy (i.e. the total of bias and variance) at iteration k is $\frac{1+\epsilon}{k} p\sigma^2$ with probability converging to 1 at a rate depending on $\epsilon \in (0, 1)$, where σ^2 is the variance of the residuals. Moreover, for specific choices of γ_1, γ_2, kSGD's objective function bias at iteration k can be shown to be $\frac{(1+\epsilon)^2}{2k^2} \|w(0) - w_*\|_2^2$ with probability converging to 1 at a rate depending on $\epsilon \in (0, 1)$, where w_* is the optimal parameter.

See also

- Coordinate descent – changes one coordinate at a time, rather than one example
- Linear classifier
- Online machine learning

References

1. Ferguson, Thomas S. (1982). "An inconsistent maximum likelihood estimate". *Journal of the American Statistical Association*. **77** (380): 831–834. doi:10.1080/01621459.1982.10477894. JSTOR 2287314.

2. Bottou, Léon; Bousquet, Olivier (2008). *The Tradeoffs of Large Scale Learning*. Advances in Neural Information Processing Systems. **20**. pp. 161–168.
3. Bottou, Léon (1998). "Online Algorithms and Stochastic Approximations". *Online Learning and Neural Networks*. Cambridge University Press. ISBN 978-0-521-65263-6
4. Kiwiel, Krzysztof C. (2001). "Convergence and efficiency of subgradient methods for quasiconvex minimization". *Mathematical Programming (Series A)*. **90** (1). Berlin, Heidelberg: Springer. pp. 1–25. doi:10.1007/PL00011414. ISSN 0025-5610. MR 1819784.
5. Robbins, Herbert; Siegmund, David O. (1971). "A convergence theorem for non negative almost supermartingales and some applications". In Rustagi, Jagdish S. *Optimizing Methods in Statistics*. Academic Press
6. Jenny Rose Finkel, Alex Kleeman, Christopher D. Manning (2008). Efficient, Feature-based, Conditional Random Field Parsing. Proc. Annual Meeting of the ACL.
7. LeCun, Yann A., et al. "Efficient backprop." Neural networks: Tricks of the trade. Springer Berlin Heidelberg, 2012. 9-48 (http://yann.lecun.com/exdb/publis/pdf/lecun-98b.pdf)
8. Díaz, Esteban and Guitton, Antoine. "Fast full waveform inversion with random shot decimation". SEG Technical Program Expanded Abstracts, 2011. 2804-2808 (http://library.seg.org/doi/abs/10.1190/1.3627777)
9. Avi Pfeffer. "CS181 Lecture 5 — Perceptrons" (PDF). Harvard University.
10. Cited by Darken, Christian; Moody, John (1990). *Fast adaptive k-means clustering: some empirical results*. Int'l Joint Conf. on Neural Networks (IJCNN). IEEE.
11. Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". *Nature*. **323** (6088): 533–536. doi:10.1038/323533a0.
12. Sutskever, Ilya; Martens, James; Dahl, George; Hinton, Geoffrey E. (June 2013). Sanjoy Dasgupta and David Mcallester, ed. *On the importance of initialization and momentum in deep learning* (PDF). In Proceedings of the 30th international conference on machine learning (ICML-13). **28**. Atlanta, GA. pp. 1139–1147. Retrieved 14 January 2016.
13. Sutskever, Ilya (2013). *Training recurrent neural networks* (PDF) (Ph.D.). University of Toronto. p. 74.
14. Zeiler, Matthew D. (2012). "ADADELTA: An adaptive learning rate method". arXiv:1212.5701.
15. Polyak, Boris T.; Juditsky, Anatoli B. (1992). "Acceleration of stochastic approximation by averaging". *SIAM J. Control and Optimization*. **30** (4): 838–855.
16. Duchi, John; Hazan, Elad; Singer, Yoram (2011). "Adaptive subgradient methods for online learning and stochastic optimization" (PDF). *JMLR*. **12**: 2121–2159.
17. Perla, Joseph (2014). "Notes on AdaGrad" (PDF).
18. Gupta, Maya R.; Bengio, Samy; Weston, Jason (2014). "Training highly multiclass classifiers" (PDF). *JMLR*. **15** (1): 1461–1492.
19. Tieleman, Tijmen and Hinton, Geoffrey (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning
20. Hinton, Geoffrey. "Overview of mini-batch gradient descent" (PDF). pp. 27–29. Retrieved 27 September 2016.
21. Diederik, Kingma; Ba, Jimmy (2014). "Adam: A method for stochastic optimization". arXiv:1412.6980.
22. Patel, V. (2016-01-01). "Kalman-Based Stochastic Gradient Method with Stop Condition and Insensitivity to Conditioning". *SIAM Journal on Optimization*. **26** (4): 2620–2648. doi:10.1137/15M1048239. ISSN 1052-6234.
23. Bertsekas, D. (1996-08-01). "Incremental Least Squares Methods and the Extended Kalman Filter". *SIAM Journal on Optimization*. **6** (3): 807–822. doi:10.1137/S1052623494268522. ISSN 1052-6234.

Further reading

- Bertsekas, Dimitri P. (1999), *Nonlinear Programming* (2nd ed.), Cambridge, MA.: Athena Scientific, ISBN 1-886529-00-0.
- Bertsekas, Dimitri (2003), *Convex Analysis and Optimization*, Athena Scientific.
- Bottou, Léon (2004), "Stochastic Learning", *Advanced Lectures on Machine Learning*, LNAI, **3176**, Springer, pp. 146–168, ISBN 978-3-540-23122-6.
- Davidon, W.C. (1976), "New least-square algorithms", *Journal of Optimization Theory and Applications*, **18** (2): 187–197, doi:10.1007/BF00935703, MR 418461.
- Duda, Richard O.; Hart, Peter E.; Stork, David G. (2000), *Pattern Classification* (2nd ed.), Wiley, ISBN 978-0-471-05669-0.
- Kiwiel, Krzysztof C. (2004), "Convergence of approximate and incremental subgradient methods for convex optimization", *SIAM Journal of Optimization*, **14** (3): 807–840,

doi:10.1137/S1052623400376366, MR 2085944. (Extensive list of references)

- Spall, James C. (2003), *Introduction to Stochastic Search and Optimization*, Wiley, ISBN 978-0-471-33052-3.

Software

- sgd (http://leon.bottou.org/projects/sgd): an LGPL C++ library which uses stochastic gradient descent to fit SVM and conditional random field models.
- CRF-ADF (http://klcl.pku.edu.cn/member/sunxu/code.htm) A C# toolkit of stochastic gradient descent and its feature-frequency-adaptive variation for training conditional random field models.
- Vowpal Wabbit: BSD licence, fast scalable learning by John Langford and others. Includes several stochastic gradient descent variants. Source repository on github (https://github.com/JohnLangford/vowpal_wabbit)

External links

- Using stochastic gradient descent in C++, Boost, Ublas for linear regression (http://codingplayground.blogspot.it/2013/05/stocastic-gradient-descent.html)

Categories: Stochastic optimization | Computational statistics | M-estimators | Machine learning algorithms | Convex optimization | Statistical approximations

---