Notes on AdaGrad

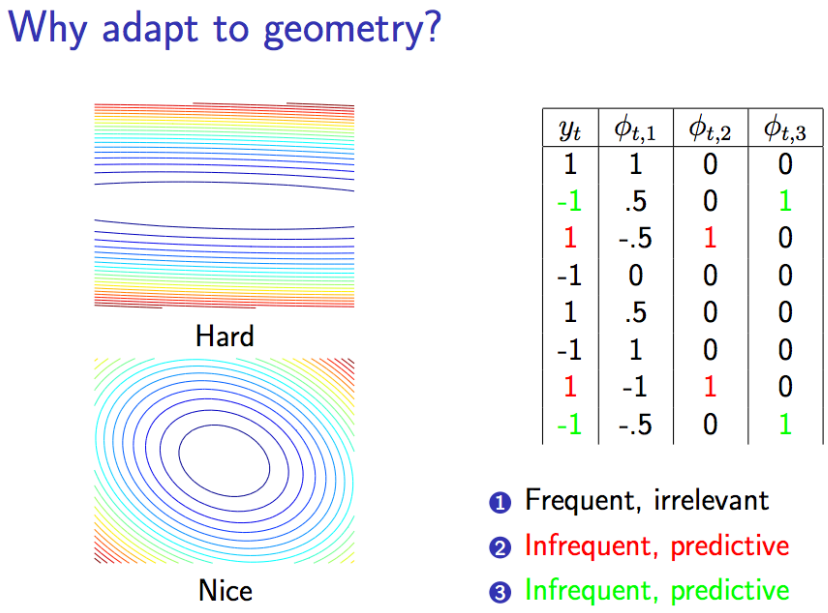
Joseph Perla

2014

1 Introduction

Stochastic Gradient Descent (SGD) is a common online learning algorithm for optimizing convex (and often non-convex) functions in machine learning today. It has many desirable benefits such as low memory usage, strong generalization abilities, and incremental results. One downside of SGD is that it is sensitive to the learning rate hyper-parameter. When the data are sparse and features have different frequencies, a single learning rate for every weight update can have exponential regret. AdaGrad is a general algorithm for choosing a learning rate dynamically by adapting to the data. AdaGrad calculates a different learning rate for every feature.

Figure 1: AdaGrad adapts to the geometry of the data. [1]



2 Algorithm

Standard stochastic gradient updates are of the following form:

$$x_{t+1} = \prod_X (x_t - \eta g_t) = argmin_{x \in X} ||x - (x_t - \eta g_t)||_2^2,$$

where we are optimizing a function from $\Re^d \to \Re$, $X = \Re^d$, $x_t \in \Re^d$, η is the learning rate, and g_t is the gradient at each time step t.

In contrast, the AdaGrad updates are of the following form:

$$x_{t+1} = \prod_X^{G_t^{\frac{1}{2}}} x_t - \eta_0 G_t^{-\frac{1}{2}} g_t,$$

where $\eta 0$ is just an initial learning rate which can in practice be 0.5 for many problems. This calculates a full covariance matrix $G_t = \sum^t \tau = 1 g_\tau g_\tau^T$ which may be intractable for large problems. We can approximate the ideal update with just the diagonal of G_t which can be computed efficiently in $O(d)$.

$$x_{t+1} = \prod_X^{diag(G_t)^{\frac{1}{2}}} x_t - \eta_0 diag(G_t)^{-\frac{1}{2}} g_t$$

3 Regret of AdaGrad

We use the standard online regret calculation of comparing the sum of the achieved losses with the sum of the best achievable loss:

$$R(T) = \sum_{t=1}^{T} f_t(x_t) - \inf_{x \in X} \sum_{t=1}^{T} f_t(x)$$

With standard stochastic gradient descent, when X is bounded ($sup_{x,y \in X} ||x - y||_2 \leq D_2$, so D_2 is the diameter of X), we achieve the following regret if we knew the optimal learning rate η in hindsight:

$$R_{SGD}(T) \leq \sqrt{2} D_2 \sqrt{\sum_{t=1}^{T} ||g_t||_2^2}. \tag{1}$$

With AdaGrad, assuming the initial x is bounded by some D_∞ from the solution (i.e. $max_t ||x * - x_t||_\infty \leq D_\infty$),

$$R_{ADA}(T) \leq \sqrt{2} D_\infty \sum_{t=1}^{d} ||g_{1:T,i}||_2. \tag{2}$$

The appendices of [2] have proofs of both the diagonal and full G_t matrix versions, in addition to versions which project the weight update onto a space $X \subset \Re^d$ and regularized versions.

4 Synthetic Examples

To gain an intuition of the algorithm, we look at some sparse synthetic datasets.

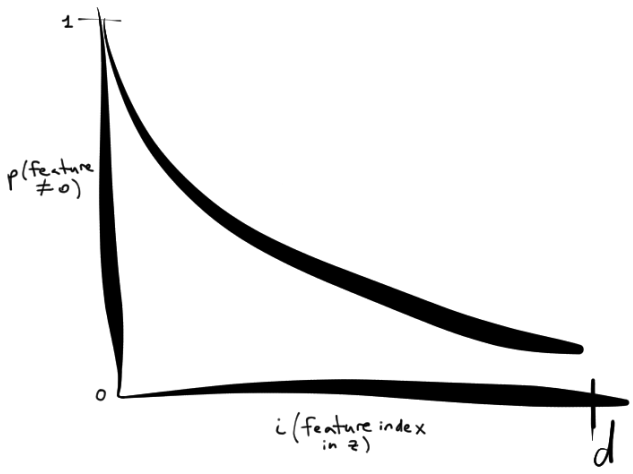
4.1 Example 1

Let the data samples be $z_t = \pm e_i$ for some i (e_i is the 0-vector with a 1 in position i). Let the labels $y_t = sign(\langle 1, z_t \rangle)$. So the perfect classifier for these data is $x* = 1 \in \Re^d$. $x_1 = 0$. Set some small ϵ. On rounds $t = 1, ..., \frac{\eta^2}{\epsilon^2}$, set $z_t = e_1$. Afterwards, choose z_t at random.

AdaGrad in this scenario suffers $d-1$ more losses after t rounds, and then has a perfect classifier. SGD has a shrunken learning rate after t rounds, $\eta/\sqrt{(}t) \leq \epsilon$ so suffers at least $d/(2\epsilon)$ which can be arbitrarily large with small epsilon. AdaGrad adapts to learning rates on a per-feature level.

4.2 Example 2

Figure 2: The data samples z_i in Example 2 are sparse, but the first indices are less sparse than the later indices, which are exponentially more sparse.



Let the data vectors be $z_t \in \{-1, 0, 1\}^d$. Assume that in each round, a feature appears with probability $p_i = min(1, ci^{-\alpha})$ for some $\alpha \in (2, \infty)$. In other words, the first few features have a high probability of being non-zero,

and the last features have an exponentially low probability of being non-zero. The data are sparse, and the sparsity per feature vary significantly. Datasets that have similar distributions are common in natural language datasets due to Zipf's law.

Take the expectation of the sum of the gradients in AdaGrad:

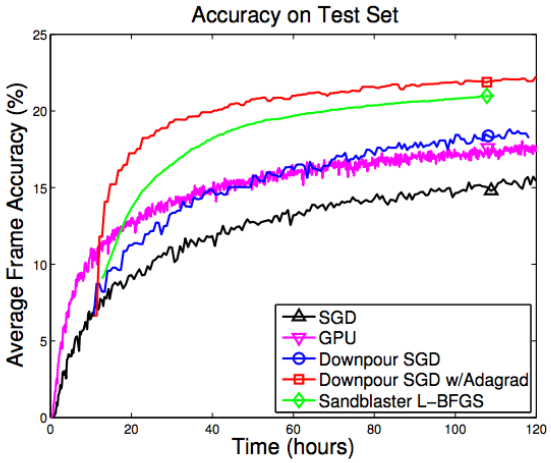
$$\mathrm{E}\sum_{i=1}^{d}||g_{1:T,i}||_2 = \sum_{i=1}^{d}\mathrm{E}\left[\sqrt{|\{t:|g_{t,i}|=1\}|}\right] \leq \sum_{i=1}^{d}\sqrt{\mathrm{E}|\{t:|g_{t,i}|=1\}|} = \sum_{i=1}^{d}\sqrt{p_i T}.$$

Then the gradient expectation is $O(\log d)$. If the domain X is a hypercube inside the 1-hypercube, then $D_\infty \leq 2$ and so based on Equation 2 the regret of AdaGrad is $O(\log d\sqrt{T})$.

In contrast, standard SGD has $D_2 = 2\sqrt{d}$ and $||g_t||_2^2 \geq 1$, so given Equation 1 its best case regret is $O(\sqrt{dT})$. AdaGrad can be exponentially better (in d) than standard SGD.

5 Real-World Data

Experiments on real data perform well. Google used AdaGrad in training its very large neural network to learn cat faces from YouTube videos efficiently.



(Dean et al. 2012)

Distributed, $d = 1.7 \cdot 10^9$ parameters. SGD and AdaGrad use 80 machines (1000 cores), L-BFGS uses 800 (10000 cores)

References

[1] Elad Hazan John Duchi and Yoram Singer. Adagrad slides. In *http://www.cs.berkeley.edu/~jduchi/projects/DuchiHaSi12_ismp.pdf*, 2011.

[2] Elad Hazan John Duchi and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *the Journal of machine Learning research*, 12:21212159, 2011.