ML Reading Group (Fall 2015) **Lecture: 6**

Large-Scale Machine Learning with Stochastic Gradient Descent

Lecturer: Kshitijh Meelu

1 Today

Recall Gradient definition and properties

Generalize Gradient Descent and Stochastic Gradient Descent

Apply Generalization to Learning Systems

Analyze Asymptotic behaviors of GD and SGD

2 Quick Summary

Author recaps decades of work on gradient descent and stochastic gradient descent, applying these methods to well-known learning algorithms and analyzing their convergence/asymptotic behavior.

3 Recall: Gradients

Given n variables, $x_1, x_2, ..., x_n$, we define the function,

$$w = f(x_1, x_2, ..., x_n)$$

Now, we define the gradient as,

$$grad(w) = \nabla w = \langle \frac{\partial w}{\partial x_1}, \frac{\partial w}{\partial x_2}, ..., \frac{\partial w}{\partial x_n} \rangle \tag{1}$$

We can also prove the following formula regarding gradients and gradient directions, which is a direct cause for their use in machine learning optimization.

Theorem 3.1. Let $w = f(x_1, x_2, ..., x_n)$. We can show that at any point $P = (x_{1_0}, x_{2_0}, ..., x_{n_0})$, on a level surface $f(x_1, x_2, ..., x_n) = c$, where c is a constant, the gradient $\nabla f|_P$ is perpendicular to the level surface.

Proof. The proof is left as an exercise in partial derivatives and multivariate calculus. Please refer to `http://ocw.mit.edu/courses/mathematics/18-02sc-multivariable-calculus-fall-2010/2.-partial-derivatives/part-b-chain-rule-gradient-and-directional-derivatives/session-36-proof/`, to view the solution. □

4 Gradient Descent and Stochastic Gradient Descent

Cost Function. Now, we use the general properties of gradients to define an algorithm to minimize cost of a learning system. We start by considering a simple supervised learning system. Because it is considered supervised, we have training examples, z, which consist of pairs (x, y). Here, x is some input and y is a scalar output.

Let $\hat{y}$ be some arbitrary prediction from input x when the actual output is y. We define $\ell(\hat{y}, y)$ to be the cost, or loss, function of predicting $\hat{y}$ when the right answer is y.

Now, we consider a family $\mathcal{F}$ of functions, $f_w(x)$, which take input x and apply this functional family, where f_w corresponds to the function parametrized by weight vector w. For this function, uniquely identified by its weight vector w, we define the cost function,

$$Q(z, w) = \ell(f_w(x), y)$$

An example of a cost function would be linear regression, where $f_w = w^T x$, and

$$Q(z, w) = ||w^T x - y||^2$$

Expected/Averaged Cost Function. To optimize the function, we seek function $f \in \mathcal{F}$ that minimizes $Q(z, w)$ averaged on examples. To do so exactly, we would like to average over the distribution $dP(z)$, which takes into account that some training examples z are more probable than others. The following is the expected risk,

$$E(f) = \int \ell(f(x), y) dP(z)$$

Because $dP(z)$ is an unknown distribution, we settle for a uniform distribution over all n training examples, giving the empirical risk,

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(x_i), y_i)$$

As an example, we give the expected cost function for linear regression:

$$E_n(f) = \frac{1}{n} \sum_{i=1}^{n} ||w^T x - y||^2$$

Gradient Descent. To minimize $E_n(f)$, we first use gradient descent (GD), where the weights w are iteratively updated based on the gradient of the averaged cost function, $E_n(f_w)$,

$$w_{t+1} = w_t - \gamma \frac{1}{n} \sum_{i=1}^{n} \nabla_w Q(z_i, w_t)$$

We note that γ here is a chose gain, or step, that defines how much the weights change per iteration. Under sufficient regularity assumptions, with sufficiently small γ, this gradient descent achieves linear convergence, which is, $-log\rho \sim t$.

Note: this version of gradient descent is considered batch gradient descent, as all training examples are batched and computed on to take one step.

Stochastic Gradient Descent. This is a simplification of gradient descent, which allows for faster computation times, especially because the machine does not have to sum over all training examples to move just *one step*. Instead, each step requires just one randomly chosen training example (normally randomization is done at the beginning). Although the algorithm is simpler, this method comes with more difficult analysis (although for SGD, especially, tons of work have been done to prove it's correctness and ability to work).

Below, we define the iterative function of stochastic gradient descent.

$$w_{t+1} = w_t - \gamma \nabla_w Q(z_t, w_t)$$

Note that studies show that convergence requires decreasing values of γ, such that both $\sum_t \gamma_t^2 < \infty$ and $\sum_t \gamma_t = \infty$ are satisfied. The speed of convergence is limited by the noisiness of the training examples. Under sufficient conditions, however, the best convergence speed comes from $\gamma_t \sim t^{-1}$. The expected residual error is then, $\mathbb{E}\rho \sim t^{-1}$.

5 Applications to Popular Learning Systems

Given the table with cost functions in the paper, deriving the stochastic gradient descent iterative function is an exercise in partial derivatives. For this reason, I plan to calculate most of the iterative functions. However, for this overview, I will go over two of them: Adaline and K-Means.

Adeline This learning system is a very slightly modified and generalized version of linear regression. We learn from the table that the cost function is,

$$Q_{adaline} = \frac{1}{2}(y - w^T \Phi(x))^2$$

Here, $\Phi(x) \in \mathbb{R}^d$ appends an $x_0 = 1$, $y = \pm 1$.

We know see that,

$$\begin{aligned} w_{t+1} &:= w_t + \gamma_t \frac{\partial}{\partial w_t} Q_{adaline} \\ &:= w_t + \gamma_t (y_t - w^T \Phi(x_t)) \Phi(x_t) \end{aligned} \tag{2}$$

K-Means The following unsupervised learning system groups all points in clusters around centroids. Rather than having z, which are training examples with actual output, we have a list of inputs, $x_1, ..., x_n$. On each repeat, the K-Means algorithm goes through all inputs, assigning each to the centroid that is closest to them. The next step is to compute a new centroid at the average of all points that are covered/clustered around the centroid. This can converge and lead to nice clusters, or it could get messy.

Here, we may define the cost function,

$$Q_{means} = \min_k \frac{1}{2}(z - w_k)^2$$

Here, z is an input, whereas the w_k is a centroid.

Then, we first find the centroid closest to the current input, z_i:

$$k* = arg\min_k (z_t - w_k)^2$$

Then, we update how many points are clustered around this specific centroid,

$$n_{k*} := n_{k*} + 1$$

Now, we derive the iterative function,

$$\begin{aligned} w_{t+1} &:= w_t + \gamma_t \frac{\partial}{\partial w_t} Q_{kmeans} \\ &:= w_t + \frac{1}{n_{k*}}(z_t - w_{k*}) \end{aligned} \tag{3}$$

6 Asymptotic Analysis

Now,we observe the errors involved with such large scale learning, and later, we will approximate certain measures across many different gradient descent algorithms.

First, note that in learning with large training sets, we will have multiple errors. Let $f^* = argmin_f E(f)$ be the pest prediction function. Because we are only observing functions in the family of functions, $\mathcal{F}$, we can denote $f^*_{\mathcal{F}} = argmin_{f \in \mathcal{F}} E(f)$ as the best function in this family. Even then, because we optimize the *empirical risk*, not the expected risk, our empirical optimum can be denoted as, $f_n = argmin_{f \in \mathcal{F}} E_n(f)$.

Furthermore, because optimization, i.e. finding the minimum in SGD, is costly, we will stop when our algorithm reach a solution $\tilde{f}_n$ that is close enough to the minimum, such that $E_n(\tilde{f}_n) < E_n(f_n) + \rho$, for some predefined ρ.

We now define and decompose the error into three separate parts:

$$\begin{aligned} \varepsilon &= \varepsilon_{app} + \varepsilon_{est} + \varepsilon_{opt} \\ &= \mathbb{E}[E(f^*_{\mathcal{F}}) - E(f^*)] + \mathbb{E}[E(f_n) - E(f^*_{\mathcal{F}})] + \mathbb{E}[E(\tilde{f}_n) - E(f_n)] \end{aligned} \tag{4}$$

Thus, given constraints on computation time, by T_{max}, and the maximal training set n_{max}, we understand that we must perform the following optimization of our optimization (meta):

$$\min_{\mathcal{F},\rho,n} \varepsilon = \varepsilon_{app} + \varepsilon_{est} + \varepsilon_{opt}$$

subject to $n \leq n_{max}$ and $T(\mathcal{F}, \rho, n) \leq T_{max}$.

From this equation we notice two cases. In small-scale learning problems, we are first contained by the number of training examples, n. Computing time, here, is not an issue. In large-scale learning problems, we are more limited by our approximation, ρ.

Moving on, from some magical mathematics to which I am not keen, we can asymptotically approximate that,

$$\varepsilon \sim \varepsilon_{app} \sim \varepsilon_{est} \sim \varepsilon_{opt} \sim \left(\frac{logn}{n}\right)^{\alpha} \sim \rho$$

Now, we can approximate the time it takes for gradient descent and stochastic gradient descent to reach some predefined excess error ε. We first show gradient descent.

Time per iteration — n, as all n training examples must be examined before the iteration can be made

Iterations to accuracy ρ — $log\frac{1}{\rho}$, which was given in the description of gradient descent

Time to accuracy ρ — $nlog\frac{1}{\rho}$, as this is Time per iteration x Iterations to accuracy ρ

Time to excess error ε — $\frac{1}{\varepsilon^{\frac{1}{\alpha}}}log^2\frac{1}{\varepsilon}$, which can be calculated from the asymptotical approximations above

Now, we show the same for stochastic gradient descent.

Time per iteration — 1, as only one must be examined before each iteration can be made

Iterations to accuracy ρ — $\frac{1}{\rho}$, which was given in the description of stochastic gradient descent

Time to accuracy ρ — $frac1\rho$, as this is Time per iteration x Iterations to accuracy ρ

Time to excess error ε — $\frac{1}{\varepsilon}$, which can be calculated from the asymptotical approximations above

We can now have further discussion on all topics presented.

7 Conclusion

This should give a general overview of the many parts to this paper. Some further areas to look into are:

- Convergence Properties: How are the convergence formulas calculated
- Asymptotic Analyses: Probabilities behind these analyses
- GD and SGD on other learning systems
- Second Order GD/SGD: Reference the following link, `http://ocw.mit.edu/courses/mathematics/18-409-topics-in-theoretical-computer-science-an-algorithmists-toolkit-fall-2009/lecture-notes/MIT18_409F09_scribe21.pdf`, pg. 4