Lecture IV: Training predictors, Part II

Marina Meilă
mmp@stat.washington.edu

Department of Statistics
University of Washington

November, 2022

Stochastic gradient methods

Examples: Linear classification with hinge loss, Perceptron

Accelerated gradient

No gradient methods: Coordinate descent

Stopping descent algorithms

Reading HTF Ch.: –, Murphy Ch.: 8.5.2-3 Stochastic gradient descent, Bach Ch.:

2em For more advanced treatment Nocedal and Wrigth.

Stochastic gradient descent (SGD) methods

[Optimization framework, minimize f over x]

SGD methods are the "cheap and slow [convergence]" methods which can however be very useful. One should not confuse "theoretically slow" with "slow in practice" and on some problems the former is true of the simpler methods but the latter is not. On other occasions, these methods perform well because they make fewer assumptions about the smoothness of the surface $f(x)$.

- Typical algorithm: steepest descent methods with diminishing step size.
- Assumption: the gradient $g^k = \nabla f(x^k)$ is computed with some error, that has 0 mean and bounded variance.
- This is the case for fitting of a model to data.

Stochastic gradient for Machine Learning: idea

[ML framework, minimize J or $\hat{L}$ over parameters, (x, y) =data, f = predictor]

- Let $\mathcal{D}_n$ be an i.i.d sample of size n from an unknown distribution.
- Denote by $\hat{L}(\theta) = -\frac{1}{n} \ln P(y^{1:n}|x^{1:n}, \theta)$ the negative log-likelihood to be minimized.
- Because the sample is i.i.d., $\hat{L}(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \ln p(y^i|x^i, \theta)$. Since f is a sum, so will be the gradient:

$$\nabla \hat{L}(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \frac{\frac{\partial p(y^i|x^i,\theta)}{\partial \theta}}{p(y^i|x^i, \theta)} \tag{1}$$

- If n is large (good from the statistical point of view) then the computation of (1) linear in n (very costly).
- The SGD idea is to set

$$d^k = \frac{\frac{\partial p(y^i|x^i,\theta)}{\partial \theta}}{p(y^i|x^i, \theta)} \tag{2}$$

  where (x^i, y^i) is randomly sampled from $\mathcal{D}_n$.
- Let P_X be the true data distribution and $\hat{P}_X$ the empirical distribution induced by the sample $\mathcal{D}_n$. Note that the direction d^k satisfies $E_{\hat{P}}[d^k] = \nabla \hat{L}$.

[Optimization framework, minimize f over x]

This example points out that in Machine Learning

- the function f and its gradient ∇f are both expensive to evaluate, because they are are sums over the potentially large sample size n
- therefore, we want to avoid not only the $\nabla^2 f$ computation, but also the ∇f computation and even the line search which entails repeated evaluations of f
- on the other hand, evaluating a noisy version of the gradient is fast, because it involves one (or a few) samples

Thus, SGD takes many imprecise steps, instead of few but very computationally demanding precise steps. It remains to see if such a method can effectively find a minimum.

Convergence of SGD

Theory It has been proved under various technical conditions[1] that stochastic gradient converges to the true optimum if

- the step sizes η^k satisfy

$$\sum_k \eta^k = \infty, \qquad \sum_k (\eta^k)^2 < \infty \quad \text{(the latter implies } \eta^k \to 0) \tag{3}$$

- and the noise variance $Var[d^k]$ is bounded.

Practically η^k's should decrease like $\frac{1}{k}$. Note however that typically in practice the decrease needs to be very slow, almost constant e.g $\frac{1}{b+k/c}$ with b, c large numbers.

[1] These results are best known under the name of Robbins-Munro theory of stochastic approximation.

Variations and practicalities

- SGD with a subsample instead of single point ("old-fashioned" SGD). At each step k, take a subsample $\mathcal{D}' \subset \mathcal{D}$ with size $|\mathcal{D}'| = n' \ll n$, compute d^k as $\frac{1}{n'} \sum_{i \in \mathcal{D}'} \nabla L(y^i, \hat{y^i})$. $\mathcal{D}'$ must be a different subset at each time.
- n' can be varied as the training progresses towards the optimum, from n' small at the start, to approach the optimum fast, to n' large near the end, to reduce variance.
- When to use "modern" SGD and when to use "old-fashioned"? Rule of thumb:
  - If you know that your function is λ-strongly convex, then use the modern algorithm with a fixed K derived from λ.
  - Otherwise, use old-fashioned.
- If the sample $\mathcal{D}$ is truly iid (if it is true e.g. that example i and $i+1$ are independent) then picking a random i can be replaced by chosing i sequentially

- On-line learning and streaming data Remarkably, SGD can naturally handle **on-line learning**, i.e. situations where data come one by one, and are not stored but discarded, after being used immediately to update the parameters.
  In on-line learning, $(x^i, y^i) \sim P_{XY}$, we would be optimizing the expected loss $L = E_P[-\ln p(Y|X, \theta)]$, and d^i would satify $E_P[d^i] = \nabla L$. Showing that the variance of d^k is bounded is no more difficult (or easy than in the finite sample case).
- Stochastic gradient and analog techniques are widely used in machine learning: training of neural networks, reinforcement learning (the TD-λ and Q-learning procedures are stochastic gradient methods), speedup of boosting.

Modern results

[ML framework, minimize J or $\hat{L}$ over parameters, (x, y) =data, f = predictor]

- Handling constraints If there are constraints $\theta \in \mathcal{A}$, we additionally assume that the *projection* $\Pi_{\mathcal{A}}\theta = \underset{\theta' \in \mathcal{A}}{\operatorname{argmin}} ||\theta - \theta'||$ can be computed efficiently (e.g. $\theta \succeq 0$).
- State of the art algorithms are very simple

Stochastic Gradient Descent (SGD)

Input $\lambda \leq \lambda_{min}\left(\nabla^2 f(x^*)\right)$, $c > 1/2$ a constant giving the step-size, [optional $\alpha \in (0, 1]$, K=total number steps]

for $k = 1, 2, \ldots K$

1. get d^k:
   - 1.1 sample a point x^i at random from $\mathcal{D}$ OR pick x^i sequentially from a random permutation of $\mathcal{D}$
   - 1.2 compute $d^k = \nabla f_\theta(x^i)$
2. update θ:

$$\theta^{k+1} \leftarrow \theta^k - \frac{c}{\lambda k} d^k \tag{4}$$

3. if $k > (1 - \alpha)K$ accumulate $\bar{\theta} \rightarrow \bar{\theta} + \theta^k$

Average

$$\bar{\theta} \leftarrow \frac{\bar{\theta}}{\alpha K} \tag{5}$$

Output $\bar{\theta}$ (or optionally θ^K)

Remarks

- steps proportional to $1/k$ and averages the last α fraction of steps.
- Empirically it was observed that no averaging, i.e. θ^K itself, has also good convergence properties, but (as expected) larger variance than $\bar{\theta}$.

Example: Linear classification with hinge loss

[ML framework, minimize J or $\hat{L}$ over parameters, (x, y) =data, f = predictor]
The following example is a classic in statistical learning. We will examine it in two formulation. The first is an example of a problem where λ is known, and the SGD theory from above applies.

Problem setting

- $y \in \{\pm 1\}$ (binary classification)
- We fit the linear classifier

$$f(x) = w^T x \tag{6}$$

- Loss function = **hinge loss**

$$L_h(y, f(x)) = \begin{cases} 0 & \text{if } yf(x) \geq 1 \\ 1 - yf(x) & \text{if } yf(x) < 1 \end{cases} = [yf(x) - 1]_- \tag{7}$$

  Define **margin** of example x

$$z = yf(x) \tag{8}$$

  Under L_h an error is penalized linearly by how far $f(x)$ is in the "wrong direction" to which we add a penalty even for correctly classified examples if the margin $yf(x)$ is below 1.
- using the hinge loss.
- Simplifying assumption (for now, will remove it when we study SVM): the data $\mathcal{D} = \{(x^i, y^i)\}_{i=1:n}$ are **linearly separable**, i.e. there exists a w^* that classifies the sample with no error. Note that in general this w^* is not unique.

Linear Support Vector Machine SGD training algorithm

The optimization problem is a regularized one:

$$J(w) = \frac{1}{n}\sum_i L_h(y^i, w^T x^i) + \frac{\lambda}{2}||w||^2 \tag{9}$$

with $\lambda > 0$ a regularization parameter chosen by the user. Note that the non-quadratic loss term is linear (with unknown slope at w^*) and therefore the function $J(w)$ is by definition λ-strongly convex.

The stochastic part of the gradient is

$$\frac{\partial L_h}{\partial w} = \begin{cases} y^i x^i & \text{if } i \text{ ``error''} \\ 0 & \text{if } i \text{ ``correct''} \end{cases} \tag{10}$$

where "correct" means that $y^i f(x^i) > 1$.

SGD for Linear SVM

Initialize with $w^0 = 0$, $\bar{w} = 0$

Iterate for $k = 1, 2, \ldots K$

1. Pick the next i in $1 : n$
2. compute direction $d^k = \lambda w^k - 1_{[i \text{ ``error''}]} y^i x^i$
3. update

$$w^{k+1} = w^k - \frac{c}{\lambda k}\left(\lambda w^k - 1_{[i \text{ ``error''}]} y^i x^i\right) = w^k(1 - c/k) + \frac{c}{\lambda k} y^i x^i 1_{[i \text{ ``error''}]} \tag{11}$$

4. average $\bar{w} \leftarrow \bar{w} + w^{k+1}$

Output $\bar{w}/K$

The Perceptron Algorithm

[ML framework, minimize J or $\hat{L}$ over parameters, (x, y) =data, f = predictor]
The second example is the PERCEPTRON algorithm from Lecture 1. We show that it is a simplified version of the previous algorithm.
There is no regularization, and the margin -1 is dropped from the hinge loss, hence we optimize

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} [y^i w^T x^i]_- \tag{12}$$

If the data is linearly separable, the minimum value of J is known to be 0; what interests is the (non-unique!) w that attains this minimum.

PERCEPTRON ALGORITHM
Initialize $w = 0$
For $k = 1, 2, \ldots$

1. pick a data point i, calculate $f(^i) = w^T x^i$
2. if i is a mistake (i.e. $y^k w^T x^i \leq 0$)

$$w \leftarrow w + y^i x^i \tag{13}$$

until no more mistakes are made.
Output w

It is easy to show Exercise Do it! that the Perceptron algorithm is a stochastic gradient descent with constant step size.
This does not satisfy the SGD convergence conditions, but there is an alternate result[2] that guarantees (a form of) convergence. Exercise It makes sense to take w^* to be the separator that maximizes γ, since this gives the tightest bound. Find what this w^* is and what the interpretation of γ should be in this case. Assume that $||x^i|| = 1$ for all i.

[2] From Lecture 1: **Proposition** The number of mistakes made by the PERCEPTRON algorithm is bounded by $\frac{1}{\gamma^2}$ where $\gamma = \min_{\mathcal{D}} \frac{|(w^*)^T x^i|}{||x^i|| ||w^*||}$ with w^* any linear separator of the data.

Accelerated gradient: the "heavy ball" method

$$x^{k+1} = x^k - \eta^k d^k + \gamma^k (x^k - x^{k-1}) \tag{14}$$

- Applies to both standard and stochastic gradient methods, i.e.

$$d^k = \begin{cases} \nabla f(x^k) & \text{gradient descent} \\ \text{noisy gradient} & \text{SGD} \\ \nabla f(x^k + \delta^k (x^k - x^{k-1})) & \text{extragradient methods} \end{cases} \tag{15}$$

- Setting the parameters
  - In the extragradient[3] methods, $\eta^k, \delta^k, \gamma^k$ are obtained by search (or knowledge about M, m)
  - For other methods fix $\gamma^k = \gamma \in (0.5, 1]$ OR use smaller γ early in the training and increase it to near 1 when the steps become smaller.
- More intuition
  - for ill conditioned problems $M \ll m$, the heavy ball "accumulates" the components of the step in the correct direction
  - for SGD, the heavy ball approximates the exact gradient

[3] Nesterov's "optimal", FISTA

Diagonally scaled SGD: AdaGrad

[ML framework, minimize J or $\hat{L}$ over parameters, (x, y) =data, f = predictor]

- In this algorithm, each component of the stochastic gradient has a different step size η_j^k, and this is "learned" as well.

AdaGrad Algorithm

Input step size η

Initialize $G_{1:n} = 0$ scaling parameters, w^0 initial parameter

- for $k = 1, 2, \ldots$
  1. pick a point i from $\mathcal{D}$
  2. compute d^k
  3. update scaling on each dimension $G_j^{k+1} = G_j^k + (d_j^k)^2$ for $j = 1 : n$
  4. take a step

$$w^{k+1} = w^k + \frac{\eta}{\sqrt{G_j^{k+1}}} d^k \tag{16}$$

  5. update the average $\bar{w}^{k+1}$
- until stopping condition

Output w (or $\bar{w}$) at last iteration

- Denote $G = \operatorname{diag}\{G_1, \ldots G_n\}$ In the algorithm, this G stands for $G^* = \sum_{t=1}^k d^t (d^t)^T$ (which would require $\mathcal{O}(n^2)$ storage). Note that the expectation $E[G^*] = kE[dd^T] = k(E[d]E[d]^T + Var(d))$. Hence the step size is asymptotically proportional to $\frac{1}{\sqrt{k}}$.[4]
- If J contains a regularization term $R(w)$ (e.g $||w||^2$), the gradient of this term should not be added to G. Note that this term of the gradient is not random, and does not depend on the data.

[4] But there exists theory to show this still is convergent.

Coordinate descent

- d^k is always one of the coordinate axes u_{i^k}. Hence $x^{k+1} = x^k + \eta_k u_{i^k}$.
- Note that line search is necessary, and that the minimum can be on either side of x^k so η_k can take negative values.

Convergence Theoretical and empirical results suggest that coordinate descent has similar convergence rate as the steepest descent (i.e linear in the best case).
While in a general case coordinate descent is suboptimal, there are several situations when it is worth considering

1. When line minimization can be done analytically. This can save one the often expensive gradient computation.
2. When the coordinate axes affect the function value approximately independently, or (in statistics) when the coordinate axes are uncorrelated. Then minimizing along each axis separately is (nearly) optimal.
3. When there exists a natural grouping of the variables. Then one can optimize one group of variables while keeping the other constant. Again, we hope that the groups are "independent", or that optimizing one group at a time can be done analytically, or it's much easier than computing the gradient w.r.t all variables simultaneously. This idea is the basis of many *alternate minimization* methods, including the well known EM algorithm.

Stopping descent algorithms

Paradigm

- What we would like is to stop when the error $f(x^*) - f(x^i)$ or $||x^* - x^k||$ is "small enough". This is possible for special classes of functions, in particular for convex functions.
- In general, we stop when some other computable quantity is "small enough", i.e smaller than a tolerance tol.

Stopping conditions for Batch algorithms (non-stochastic)

- What **not** to do:
  - stop when $k = 100$ (or any other pre-set number K)
  - stop when $||\nabla f(x^k)|| < tol$ Exercise Why?
  - set $tol < \sqrt{\varepsilon_{machine}} \approx 10^{-8}$
- What to do:
- The "poor man's" stopping condition: $\left|1 - \frac{f(x^{k+1})}{f(x^k)}\right| < tol$[5]
- The "pro's" stopping condition: Newton step $= ||\nabla^2 f(x^k)^{-1} \nabla f(x^k)|| < tol$.
  Note: don't compute it at every step (unless you are actually running Newton method), but only once in a while, depending on n and what descent algorithm you are using.

[5] The $||$ are not necessary if the method you use guarantees $f(x^{k+1}) < f(x^k)$ but this is not always the case. Note also that this fails if $f(x^k) = 0$ or changes sign. But it works well for loss functions as they are always positive.

Stopping SGD

[ML framework, minimize J or $\hat{L}$ over parameters, (x, y) =data, f = predictor]

- L is strongly convex, and lower bound on λ known. (Hence you are using "modern" SGD.)
  - Fix K in advance by using the theorem $E[||\theta^* - \theta^k||^2] \leq \frac{c' G^2}{\lambda^2 K^2}$ (c' is a function of c) and setting $tol^2 > \frac{c' G^2}{\lambda^2 K^2}$
- otherwise (old-fashioned SGD, with $n' = 1$ or larger)
  - every M iterations, where M is large enough, test if $\frac{||\bar{\theta}^k - \bar{\theta}^{k-M}||}{||\bar{\theta}^k||} < tol$