IE510 Applied Nonlinear Programming

Lecture 8: Stochastic Gradient Methods

Ruoyu Sun

Mar 2018. Week 8 Thur to Week 9 Thur.

Questions for Lecture 7 on Coordinate Descent

- **Q1**: What is the essential idea for CD?

Candidate answers:
- simpler subproblems (GD)
- alternating

Answer: Decompose (small)

For "big" data problems, need to cut into small pieces.

- **Q2**: When does CD fail to converge? When does it converge?

- **Q3**: Why is CD better than GD for some problems?

Questions for Lecture 7 on Coordinate Descent

- **Q1**: What is the essential idea for CD?

  Decompose (small)

- **Q2**: When does CD fail to converge? When does it converge?

  Failure case: nonsmooth

  Converge case: Smooth + per-block strict convexity

- **Q3**: Why is CD better than GD for some problems?

  Faster (bigger stepsize) (combining with efficiency trick)
  + smaller memory

Review of Lecture 7 on Coordinate Descent

- **Variants**: Update order, block size, subproblem solver
  - cyclic coordinate descent (C-CD)
  - Randomized block coordinate gradient descent (R-BCGD)

  Advantages:
  - Advantage 1: Faster (bigger stepsize)
  - Advantage 2: Smaller memory
  - Advantage 3 (if exact versions): no stepsize tuning
- **When converge**? (every limit point is stationary)
  - Smooth + per-block strict convexity (if not, construct such auxiliary subproblem)
  - Diverge if general nonsmooth problem

e.g. of $f(x_i) + \frac{1}{2}\|x_i - x_i^{old}\|^2$ is strict convex, then apply to CD variant that updates $x_i \leftarrow \arg\min_{x_i} f(x_i) + \frac{1}{2}\|x_i - x_i^{old}\|^2$

Today

- This Lecture: Stochastic Gradient Methods
- After this lecture, you will be able to
  - define stochastic gradient methods (SGD)
  - identify problems that are suitable for SGD
  - tell the difference of SGD and GD in terms of convergence theory
  - tune stepsize to make SGD converge (for some problems)

Outline

Motivation: Many Samples

- Linear regression $\min_w \sum_i (y_i - \langle w, x_i \rangle)^2$.
- GD: each iteration takes time $O(nd)$, space $O(nd)$
- Big data age: n, d can be large
- CD: each iteration updates one feature w_i for all n samples
  - Time: $O(n)$
  - Space: $O(n)$

Motivation: Many Samples

- What if $n = 10^5$, $d = 200$? Time and space still large

- Solution: each iteration utilizes one sample x_i for all d features

- Per-sample loss: $f_i(w) = (y_i - \langle w, x_i \rangle)^2$, $w, x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$.

- Can you solve w?

  No.

- We can update w by performing a gradient step:

$$w \leftarrow w - \alpha \nabla f_i(w).$$

Motivation: Many Samples

- What if $n = 10^5$, $d = 200$? Time and space still large
- Solution: each iteration utilizes one sample x_i for all d features
- Per-sample loss: $f_i(w) = (y_i - \langle w, x_i \rangle)^2$, $w, x_i \in \mathbb{R}^d, y_i \in \mathbb{R}$.
- Can you solve w?

  No. (more precisely: you can solve it exactly but that's not good)

  Don't go that far to solve it exactly. One sample only changes your decision variables w a bit.
- We can update w by performing a gradient step:

$$w \leftarrow w - \alpha \nabla f_i(w).$$

SGD for Finite Sum

- Consider an optimization problem

$$\min_{w\in\mathbb{R}^d} F(w) \triangleq \sum_{i=1}^{n} f_i(w),$$

where f_i is differentiable. (may or may not be convex)

- **SGD** (Stochastic Gradient Method): at each iteration t ($t=1,2,\cdots,\infty$), randomly pick a sample i, then update

($\nabla f_i(w^t)$: gradient at sample i)

$$w^{t+1} = w^t - \alpha_t \nabla f_i(w^t).$$

- Remark: SGD is not necessarily a descent method; so in optimization field, people call SG method. But following convention in machine learning, we call SGD in the lecture.

("on average", will decrease. $F(w^{t+1}) = w^t - \alpha_t \nabla F(w^t)$.)

Variants of SGD for Finite Sum

Choices: update order, block size, ~~subproblem update~~
GD choices (stepsize rule + momentum) (not a choice for SGD, typically)

- Update order: cyclic, random permutation
  - Incremental gradient method (in textbook)
- Block size: mini-batch SGD

SGD = randomized incremental
incremental = cyclic of SGD

Block CD (BCD)
mini-batch SGD
different names, similar idea

$$w \leftarrow w - \alpha_t \sum_{k \in B_i} \nabla f_k(w).$$

- SGD with momentum

$$w \leftarrow w - \alpha_t \nabla f_i(w) + \beta(w - w^{\text{old}})$$

- Stepsize? Not that straightforward...

More on Update Order

* **Best order:**
  random permutation

* Update order often only affect "convergence speed": if you use small enough stepsize under same stepsize rule, they either all converge or not converge.
  (except recent finding in ADMM: the update order matters; see [Sun, Luo, Ye '15]

  However, faster speed means
  "wider range of stepsize s.t. the method converges".

* Question: Pick coordinate + sample together,
  e.g. pick feature i and sample k, update $x_i \leftarrow x_i - \alpha \frac{\partial f_k(x)}{\partial x_i}$?

  Yes, you can do this; but when it works is unclear.
  Some papers on this.

Example 1: Empirical Loss Minimization

We have seen linear regression: $\min_w \sum_i (a_i^T w - b_i)^2$.

In machine learning, a common problem is

$$\min_w \sum_i f_i(w) = \min_w \sum_i l(y_i, \phi_w(x_i)).$$

$\phi_w(x_i)$: prediction for sample i, a function of data x_i, depends on parameter w

Here $f_i(w) = l(y_i, \phi_w(x_i))$, where l is a loss function.

- Linear prediction: $\phi_w(x_i) = w^T x_i$
  - Linear regression: $l(y, z) = \|y - z\|^2$
  - Logistic regression: $l(y, z) = -log(1 + \exp(-yz))$
  - Hinge loss: $l(y, z) = \max(0, 1 - yz)$
- Neural-networks: $\phi_w(x_i) = W_K \sigma(W_{K-1} \sigma(\dots w_2 \sigma(W_1 x_i)))$.

σ: nonlinear function

Example 2: Matrix Factorization

Consider the problem

$$\min_{X,Y} \|M - XY^T\|_F^2.$$

Simpler case:

$$\min_{x,y \in \mathbb{R}^n} \|M - xy^T\|_F^2.$$

We showed CD can solve it. How about SGD?

Rewrite as $\min_{\{x_i\},\{y_j\}} \sum_{i,j} (M_{ij} - x_i y_j)^2$.

Write SGD for the problem:

Extension: solve $\min_{\{x_i\},\{y_j\}} \sum_{i,j} l(M_{ij}, x_i y_j)$, for some l.

Example 2: Matrix Factorization

Consider the problem

$$\min_{X,Y} \|M - XY^T\|_F^2.$$

We showed CD can solve it. How about SGD?

Rewrite as $\min_{\{x_i\},\{y_j\}} \sum_{i,j}(M_{ij} - x_i y_j)^2$,

Write SGD for the problem:

$$\begin{pmatrix} x_1 \\ \vdots \\ x_m \\ y_1 \\ \vdots \\ y_n \end{pmatrix} \leftarrow \begin{pmatrix} x_1 \\ \vdots \\ x_m \\ y_1 \\ \vdots \\ y_n \end{pmatrix} - \begin{pmatrix} \partial f_{ij}/\partial x_1 \\ \vdots \\ \partial f_{ij}/\partial x_m \\ \partial f_{ij}/\partial y_1 \\ \vdots \\ \partial f_{ij}/\partial y_n \end{pmatrix},$$

Extension: solve $\min_{\{x_i\},\{y_j\}} \sum_{i,j} l(M_{ij}, x_i y_j)$, for some l.

General framework:

$\min_k \sum_k f_k(w)$,

SGD: $w^+ = w - \alpha \nabla f_k(w)$

What is $f_k(w)$?

$f_{ij}(w) = (M_{ij} - x_i y_j)^2$

What is w?

$w = (X, Y)$

$= (x_1, \cdots, x_m, y_1, \cdots, y_n)$.

$i = 1, \cdots, m$; $j = 1, \cdots, n$.

equivalent to $\begin{pmatrix} x_i \\ y_j \end{pmatrix} \leftarrow \begin{pmatrix} x_i - \alpha(M_{ij} - x_i y_j) y_j \\ y_j - \alpha(M_{ij} - x_i y_j) x_i \end{pmatrix}$

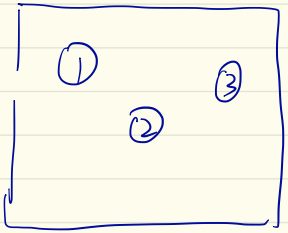
difference

compare to CGD:

$$\begin{cases} x_i^+ = x_i - \alpha(M_{ij} - x_i y_j) y_j \\ y_j^+ = y_j - \alpha(M_{ij} - x_i^+ y_j) x_i^+ \end{cases}$$

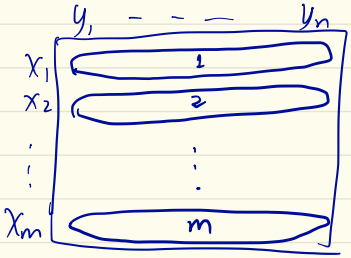
Different versions of SGD (useful in practice).

SGD version 1:



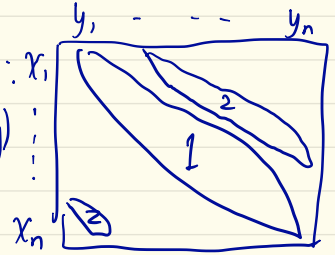
* every iteration sample M_{ij}.
* $f_k(w) = (M_{ij} - x_i y_j)^2$

SGD version 2:



* every iteration use samples
  $M_{i,1}, M_{i,2}, \dots, M_{in}$
* Component function: (for fixed i)
  $f_k(w) = \sum_{j=1}^{n} (M_{ij} - x_i y_j)^2$

SGD version 3: (assume $m=n$ for simplicity) (much faster)



* each iteration use samples
  $M_{1,k}, M_{2,k+1}, \dots, M_{n,k+n}$
  (let $M_{n,n+1} = M_{n,1}$, etc.)
* component function
  $f_k(w) = \sum_{j=1}^{n} (M_{j,k+j} - x_j y_{k+j})^2$.

Similar ideas appear in Jelly Fish (Recht), DSCOVR (Lin Xiao, Microsoft).

Example 3: Regularizers

Typically, we add regularizers

$$\min_w \frac{1}{n}\sum_i f_i(w) + \lambda\|w\|^2.$$

How to write SGD for it?

Version 1. $\min_w \frac{1}{n}\sum_{i=1}^n (f_i(w) + \lambda\|w\|^2)$ → practical choice

$g_i(w)$

pick i, $w \leftarrow w - \alpha \nabla g_i(w)$

Version 2. $\min_w \frac{1}{n}\sum_{i=1}^n f_i(w) + \underbrace{\lambda\|w\|^2}_{\triangleq f_{n+1}(w)}$ → for fun: this is also a valid partion of component functions.

See more examples beyond machine learning at [Ber10].

Version 3. $\frac{1}{n}(f_1(w) + n\lambda\|w\|^2) \triangleq g_1(w)$, $f_2(w)$, $\vdots$, $f_n(w)$ → again, for fun.

Preliminary Comparison of SGD and CD

Same class of methods: decomposition and solve subproblems iteratively.

Big data optimization: decomposition + gradient.

Similarities of SGD and CD

- Both decomposition
- Similar variants: cyclic/randomized, block, etc.

Differences of SGD and CD

- Application range:
- Efficiency
- Convergence theory

Preliminary Comparison of SGD and CD

Same class of methods: decomposition and solve subproblems iteratively.

Big data optimization: decomposition + gradient.

Similarities of SGD and CD

- Both decomposition
- Similar variants: cyclic/randomized, block, etc.

Differences of SGD and CD

- Application range:
- Efficiency { $n > d$: SGD; $d > n$: CD. (for some problems)
- Convergence theory

Other differences:
- update method: exact v.s GD (but CGD exists, so CD-methods can use gradient update as well)
- Descent v.s non-descent (CD) (SGD)
- More samples: use SGD; More features: use CD
- Online case: SGD (physical constraint: data come one by one)

Outline

Example of 2 samples

Example: Consider minimizing $f(x) = \frac{1}{2}(x-1)^2 + \frac{1}{2}(x+1)^2$. Given in [Luo91]

- Clearly, f is strongly convex and $x^* = 0$.
- Pick any x^0. The incremental gradient method is

$$x^r(2) = x^r(1) - \alpha(x^r(1) - 1)$$
$$x^{r+1}(1) = x^r(2) - \alpha(x^r(2) + 1),$$

where $x^r(i)$, i = 1,2, denotes the iterate just before the i-th component in the r-th cycle.

Example of 2 samples

Example: Consider minimizing $f(x) = \frac{1}{2}(x-1)^2 + \frac{1}{2}(x+1)^2$. Given in [Luo91]

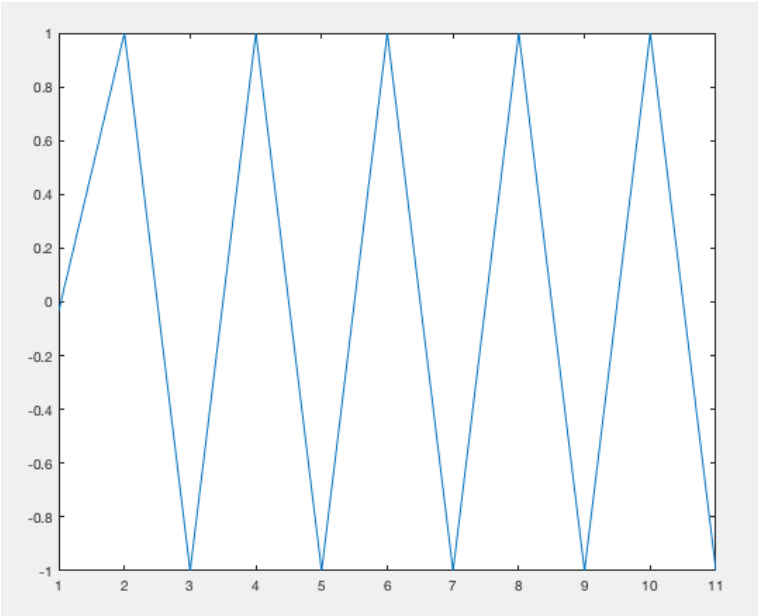
- Clearly, f is strongly convex and $x^* = 0$.

- Pick any x^0. The incremental gradient method is

$$x^r(2) = x^r(1) - \alpha(x^r(1) - 1)$$
$$x^{r+1}(1) = x^r(2) - \alpha(x^r(2) + 1),$$

  where $x^r(i)$, i = 1,2, denotes the iterate just before the i-th component in the r-th cycle.

Example: Plots of Iterates



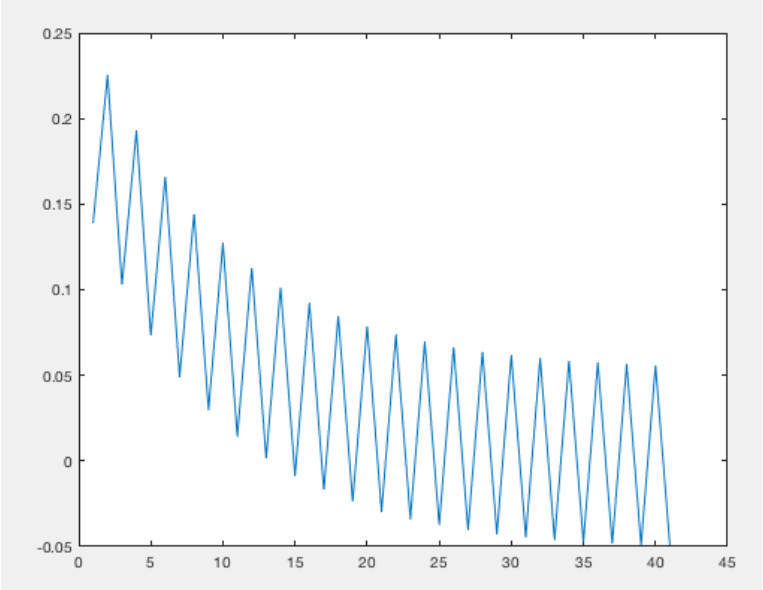


Figure: Left: $\alpha = 1$; Right: $\alpha = 0.1$

What do you observe? Explain by theory?

Limit Points

- Write down a single-recursion formula:

$$x^{r+1}(1) = (1-\alpha)^2 x^r(1) - \alpha^2.$$

- For $0 < \alpha < 1$, the sequence $x^r(1) \to \frac{-\alpha}{2-\alpha} = x_\alpha(1)$
- similarly, $\lim_{r\to\infty} x^r(2) = x_\alpha(2) = \frac{\alpha}{2-\alpha}$.
- **Lesson 1**: for fixed step size $\alpha < 1$, the sequence of iterates will oscillate
  - oscillate between two limiting points $x_\alpha(1)$ and $x_\alpha(2)$.
- Notice that

$$|x_\alpha(1) - x^*| = |x_\alpha(2) - x^*| = O(\alpha).$$

  Lesson 2: The final error is proportional to the stepsize!
- This suggests when _____, both $x_\alpha(1)$ and $x_\alpha(2)$ might converge to $x^* = 0$.

Limit Points

- Write down a single-recursion formula:

$$x^{r+1}(1) = (1-\alpha)^2 x^r(1) - \alpha^2.$$

- For $0 < \alpha < 1$, the sequence $x^r(1) \to \frac{-\alpha}{2-\alpha} = x_\alpha(1)$
- similarly, $\lim_{r\to\infty} x^r(2) = x_\alpha(2) = \frac{\alpha}{2-\alpha}$.
- **Lesson 1**: for fixed step size $\alpha < 1$, the sequence of iterates will oscillate
  - oscillate between two limiting points $x_\alpha(1)$ and $x_\alpha(2)$.
- Notice that

$$|x_\alpha(1) - x^*| = |x_\alpha(2) - x^*| = O(\alpha).$$

  Lesson 2: The final error is proportional to the stepsize!
- This suggests when _____, both $x_\alpha(1)$ and $x_\alpha(2)$ might converge to $x^* = 0$.

Graph Illustration

Graph Illustration: Multiple quadratic functions.

Graph Illustration

Graph Illustration (2-dim)

- fast convergence zone: when far away
- confusion zone: when close

Extreme case: all f_i are the same

Graph Illustration

2-D example: fast convergence zone and confusion zone

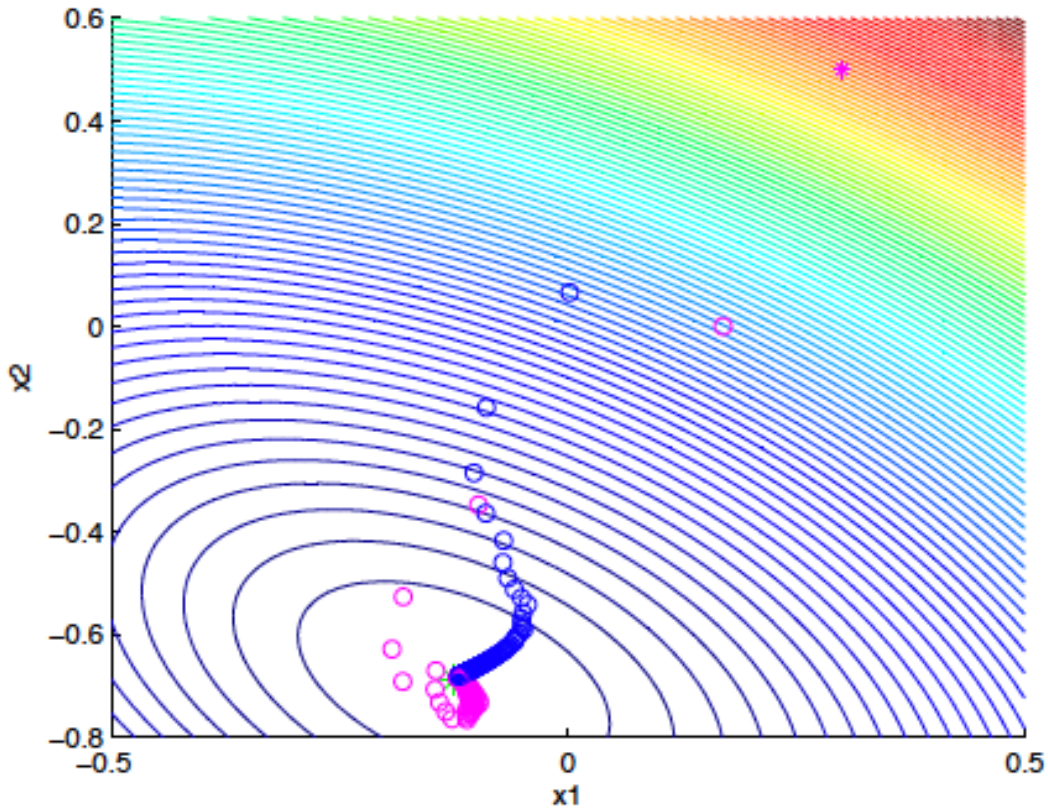
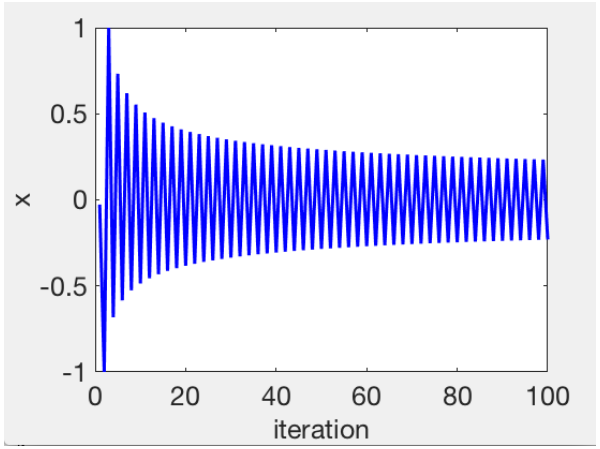


Figure: From

Varying Stepsize



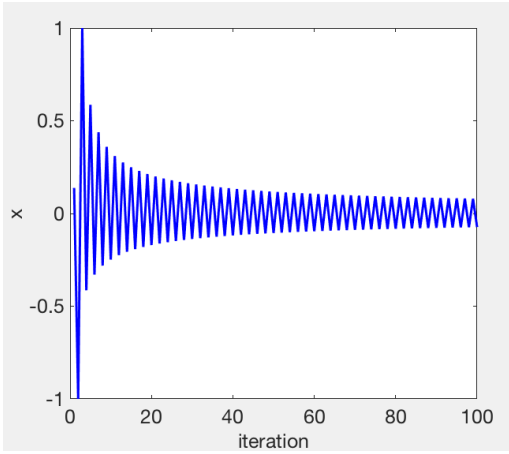
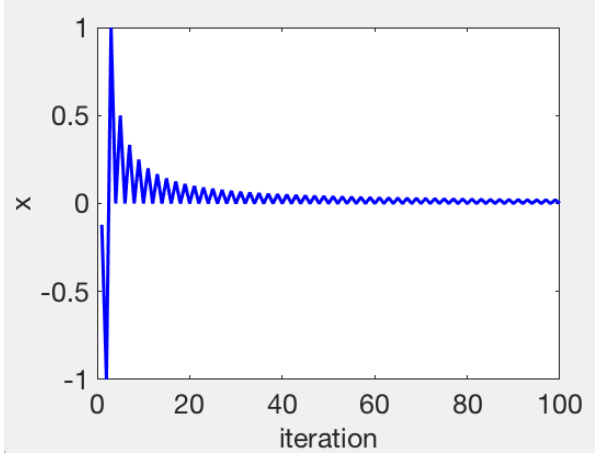


Figure: Left: $\alpha_t = 1/t^{0.25}$; Right: $\alpha_t = 1/\sqrt{t}$



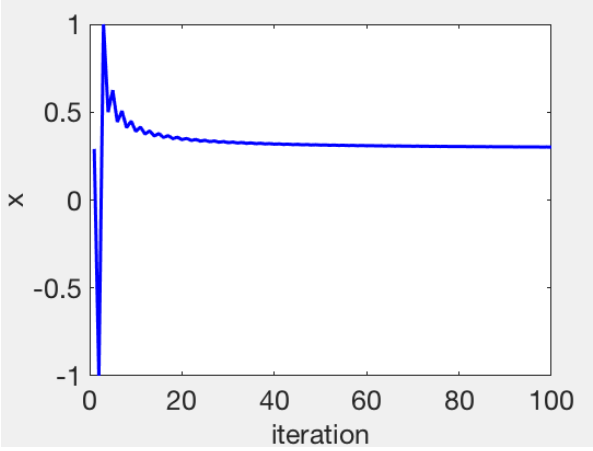


Figure: Left: $\alpha_t = 1/t$; Right: $\alpha_t = 1/t^2$

Stepsize schedule

- Observation: $\alpha_t = 1/t$ works; but other α_t does not.
- Dynamically decreasing step sizes $\alpha_t \to 0$:
  - too slow $\Rightarrow$ $\{x^t(1)\}$ and $\{x^t(2)\}$ still converge to two different limit points.
  - too fast $\Rightarrow$ the iterates will not reach x^*
- What is the condition on stepsize α_t ? We know
  - $\sum_t 1/t = \infty$
  - $\sum_t 1/t^2 < \infty$
  - In fact, $\sum_t 1/t^\beta < \infty$ iff $\beta > 1$

Outline

Convergence Results Types

- Similar to GD/CD, two major types of results
  - Convergence (possibly non-convex)
  - Convergence rate (convex and strongly convex)
- Other factors: update order, stepsize rule, etc.
- We emphasize one thing for the rigorous convergence result: stepsize rule

Convergence Result (Non-convex)

Assumption 1 (Lipschitz gradient): F is continuous differentiable and

$$\|\nabla F(x) - \nabla F(y)\| \leq L\|x - y\|.$$

Assumption 2 (gradient bound) For some constant $C, D > 0$,

$$\|\nabla f_i(x)\| \leq C + D\|f(x)\|, \forall i.$$

Assumption 3 (stepsize) The stepsize satisfies

$$\sum_{t=1}^{\infty} \alpha_t = \infty, \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty. \tag{1}$$

Proposition 8.1 Suppose we use incremental gradient method $w^{t+1} = w^t - \alpha_t \nabla f_i(w^t)$ to solve $\sum_{i=1}^n f_i(w)$. Under Assumption 1,2,3, for either of the following two update order

- cyclic rule,
- random permutation rule,

every limit point of the sequence is a stationary point.

Stepsize Requirement

- **Claim**: $1/t^\beta$ for $0.5 < \beta \leq 1$ satisfies (1)
  - Fact: for $c \geq 0$, $\sum_{t=1}^{\infty} 1/t^c = \infty$ iff $0 \geq c \leq 1$
- How about $\beta = 0.5$?

  No. $\sum_t 1/t^{2\beta} = \sum_t 1/t = \infty$.
- Is $0.5 < \beta \leq 1$ really necessary for convergence?
  - Not clear from simulation (converge too slow, 10^6 iterations not enough to tell)
  - $\beta = 0.4$ or $\beta = 1.1$ good enough for this 1-dim problem (even if not converging, error is small)

Stepsize Requirement

- **Claim**: $1/t^\beta$ for $0.5 < \beta \leq 1$ satisfies (1)
  - Fact: for $c \geq 0$, $\sum_{t=1}^{\infty} 1/t^c = \infty$ iff $0 \geq c \leq 1$
- How about $\beta = 0.5$?

  No. $\sum_t 1/t^{2\beta} = \sum_t 1/t = \infty$.
- Is $0.5 < \beta \leq 1$ really necessary for convergence?
  - Not clear from simulation (converge too slow, 10^6 iterations not enough to tell)
  - $\beta = 0.4$ or $\beta = 1.1$ good enough for this 1-dim problem (even if not converging, error is small)

About This Result

- SGD converges to stationary points (in the sense of “every limit point is stationary”) under:
  - Proper diminishing stepsize choice (1)
  - Lipschitz gradient
  - Gradient error/variance bounded
- This is one of the most general convergence result for SGD
  - For randomized rule, similar convergence results hold
- However, Lipschitz gradient is a big assumption
  - Technically, this result does not apply to deep learning or matrix factorization
  - Adding regularizer might make it work, but the analysis is probably not clean

Practical Tricks

- Diminishing stepsize rule (1) is a "safe" choice
  - Use $\alpha/(t+\Delta)$, where Δ is a big constant (decay slower)
- Popular schedule: "diminish when stuck"
  - use constant stepsize for a while,
  - then diminish (divide by 2 or 10) after a while or whenever stuck
- Another trick: return weighted average of iterates
  - Use $1/t^{0.75}$ as stepsize and $\frac{1}{\max\{1,t-d,t-n\}}$ as weights [Bot12]
- Yet another trick: to decide α, use 5% of the data; bisection to find best α

Learning Rate Schedule: Mystery

- Learning rate schedule is critical for practical performance
  - Once a headache in deep learning
  - Lots of recent papers on this issue
  - One example: tuning stepsize up and down (ICLR'18).
- It's amazing SGD can be made to work for neural-nets now
  - That doesn't mean you can make it work
  - Different data? non-CNN? LSTM + attention?
- Why hard? Mix eigenvalue effects + SGD effects + non-convex effects

Mystery of Stepsize Schedule

- Eigenvalue effects: constant stepsize not the best choice even for linear regression
- SGD effects: different phases; how to diminish?
- non-convex effects: even GD for $(1 - w_1 w_2)^2$, behavior not understood
- What's more? Momentum effects

Outline

Convergence Rates for Convex Case

Default assumptions: Lipschitz gradient

Assume F is convex or strongly convex. + bounded gradient error

For diminishing stepsize satisfying (1), two sets of results :

- For strongly convex, $O(1/\epsilon)$ to achieve error ϵ, need $O(\frac{1}{\epsilon})$ iterations.
  Lecture 2C: GD (const. stepsize) $O(k \cdot \log\frac{1}{\epsilon})$. linear.
- For non-strongly convex, $O(1/\epsilon^2)$ sublinear
  GD: (convex) $O(\frac{1}{\epsilon})$

For constant stepsize, two sets of results :

$\epsilon = 0.001$, $\frac{1}{\epsilon} = 1000$ GD, $\frac{1}{\epsilon^2} = 10^6$ SGD

- For strongly convex, $O(\log 1/\epsilon)$ time to achieve $\epsilon + O(\alpha)$ error
- For non-strongly convex, $O(1/\epsilon)$ time to achieve $\epsilon + O(\alpha)$ error

"similar" to GD

worse than GD.

See [Bot16] for formal results.

A missing message

- Bottou et al. 2016 “we omit these complications (of almost sure convergence) since ... they do not provide significant additional insights into the forces driving convergence of the method”
- These theoretical results usually miss an important message.
- They show why SGD is worse than GD, but not why SGD is better
    - SGD is widely used not because $1/\epsilon$ rate or $O(\alpha)$ error...
- but because SGD is faster than GD, in certain sense...
- SGD is faster since **bigger stepsize allowed**
    - Same reason as why CD is faster than GD

A missing message

- Bottou et al. 2016 “we omit these complications (of almost sure convergence) since ... they do not provide significant additional insights into the forces driving convergence of the method”
- These theoretical results usually miss an important message.
- They show why SGD is worse than GD, but not why SGD is better
  - SGD is widely used not because $1/\epsilon$ rate or $O(\alpha)$ error...
- but because SGD is faster than GD, in certain sense...
- SGD is faster since **bigger stepsize allowed**
  - Same reason as why CD is faster than GD

To be precise, they may have shown why SGD is better if you check constants carefully, but usually not emphasized

A missing message

- Bottou et al. 2016 “we omit these complications (of almost sure convergence) since ... they do not provide significant additional insights into the forces driving convergence of the method”
- These theoretical results usually miss an important message.
- They show why SGD is worse than GD, but not why SGD is better
  - SGD is widely used not because $1/\epsilon$ rate or $O(\alpha)$ error...
- but because SGD is faster than GD, in certain sense...
- SGD is faster since **bigger stepsize allowed**
  - Same reason as why CD is faster than GD

Extreme Case that SGD is Fast

- Consider linear regression $\min_w F(w) = \sum_{i=1}^n \frac{1}{n} f_i(w)$, where $f_i(w) = x_i^T w - y_i$.

- Suppose $x_1 = x_2 = \cdots = x_n$, $y_1 = \cdots = y_n$, i.e., all data are identical. Then $F(w) = f_i(w)$.

$\nabla f_i = \nabla F$

- So one iteration of SGD $w \leftarrow w - \beta_t \nabla f_i(w)$ is equivalent to one iteration of GD $w \leftarrow w - \alpha \nabla F(w)$ if $\beta_t = \alpha$; while cost is $1/n$-th

1 ite of SGD: $\nabla f_i(w)$

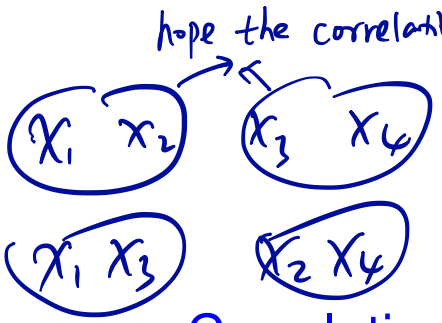
1 ite of GD: $\nabla F = \frac{1}{n} \sum_i \nabla f_i(w)$. n times more computation than SGD

- **Another view**: one epoch of SGD has n updates, each with stepsize α, thus "epoch-stepsize" $n\alpha$, n times larger than GD

one epoch: one pass of all data.

$w^1 \xrightarrow{\alpha} w^2 \xrightarrow{\alpha} w^3 \to \cdots \xrightarrow{\alpha} w^n$ one epoch of SGD, epoch stepsize $n\alpha$.

$n\alpha$



Correlation is Good for SGD

- Correlation between mini-batches larger $\Rightarrow$ faster
  - **Practical tip**: Shuffle data, s.t., similar data are not in same mini-batch
  - e.g. binary classification, each mini-batch has balanced positive/negative samples

or: do clustering of data

Cluster 1 x_1 x_2 x_3 x_4 Cluster 2 x_5 x_6 x_7 x_8

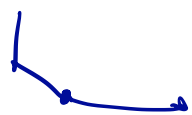
Pick (x_1, x_2, x_5, x_6) as mini-batch

(x_3, x_4, x_7, x_8) as mini-batch

- Same epoch-stepsize means no speedup
  - If picking the same epoch-stepsize for SGD and GD, i.e., the per-iteration stepsize $\alpha_t = \alpha/n$, then SGD is as slow as GD
  - after shuffling, use bigger stepsize
- Remark: pay attention to scaling of $1/n$
  - average $F(w) = \sum_{i=1}^n \frac{1}{n} f_i(w)$ v.s. the sum $F(w) = \sum_{i=1}^n f_i(w)$.
  - "effective stepsize" for the two problems differ by a factor of n

Bigger Stepsize of SGD

- Usually, the "epoch-stepsize" of SGD is 1 to n times bigger than GD
- The speed often scales linearly with the "epoch-stepsize", so SGD is often 1 to n times faster than GD (in early stage)
- Of course, a precise comparison is difficult due to:
  - "Confusion zone" of SGD
  - Multi-stage behavior
  - Nonconvexity



Rigorous Explanation of Bigger Stepsize? (can skip in class)

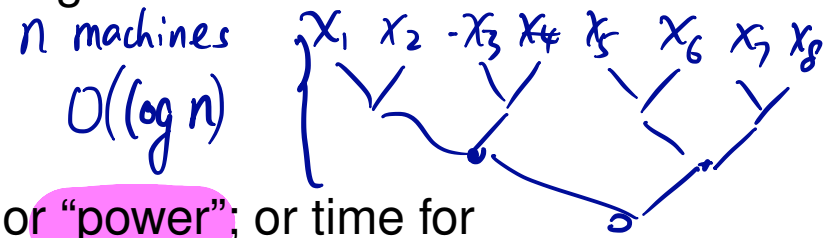
- For a rigorous treatment of "bigger stepsize" for CD in quadratic case, see IE598 Post on "Coordinate descent" acceleration ratio $\frac{\lambda_{max}}{\lambda_{average}} \in [1, n]$
- For least squares, CD = Kaczmarz method = SGD with optimal stepsize
- SVRG/SAGA: recently proposed faster versions of SGD
  - SDCA, Finito
  - Converges even when constant stepsize (variance goes to 0) variance reduction techniques
  - Achieve same rate as CD for least squares when $n = d$
    - Rigorously speaking, need to take transpose of the matrix. SVRG/SAGA depends on row norm, and R-CD depends on column norm

even experts may not know this

- Due to these relations, one can apply the discussion of stepsizes for CD to SVRG/Kaczmarz method (better versions of SGD)

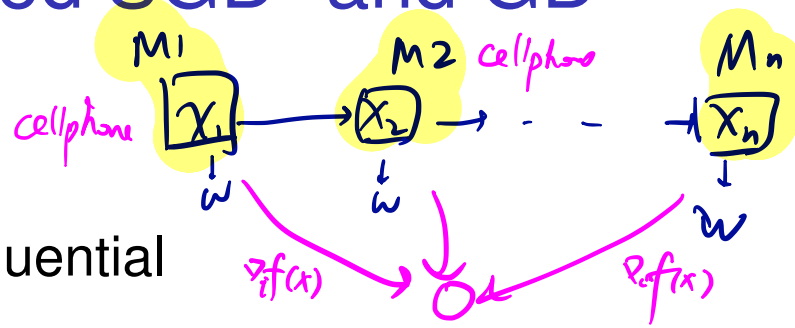
Parallel Computation

- All the previous comparison (SGD/CD is faster than GD) is based on an assumption: sequential computation.
- For example, we say $x_1 + x_2 + \cdots + x_n$ takes "time" $O(n)$
  - However, it takes time $O(\log n)$ if using trees

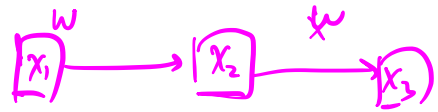


- "time" may refer to:
  - **Meaning 1**: number of operations; or "power"; or time for sequential computation
  - **Meaning 2**: time for (possibly) parallel/distributed computation

"distributed SGD" and GD



- SGD, by original definition, is sequential
- What if n nodes update $w \leftarrow w - \alpha \nabla f_i(w)$ at the same time?
  - n copies of w
  - averaging them? it becomes GD $w \leftarrow w - \alpha \frac{1}{n} \sum_i f_i(w)$
- Naive parallel SGD = GD
- Assume d is large, n is huge, so one machine only stores one sample
- Communication cost is high... fully parallel hard



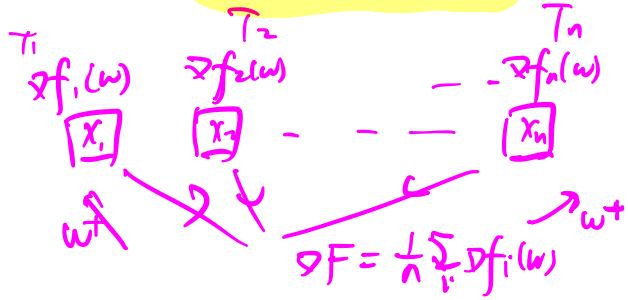
Comparing SGD and GD

A simple calculation comparing SGD and GD

$n = 20, 30, 100$, etc.

or a mini-batch of samples; this does not affect the analysis.

- Assuming n machines, each has a sample x_i; one control center
- Suppose computing $\nabla f_i(w)$ takes time T_i
- one epoch of (sequential) SGD takes time $\sum_i T_i + C$
- one epoch of GD takes time $\max\{T_i\} + 2C$;
  - takes n machines $\max\{T_i\}$ time to compute all gradients $\nabla f_i(w)$
  - sum up the gradients + update w at control center + inform all machines the new w
  - suppose communication takes time C



slowest $\max\{T_i\}$.

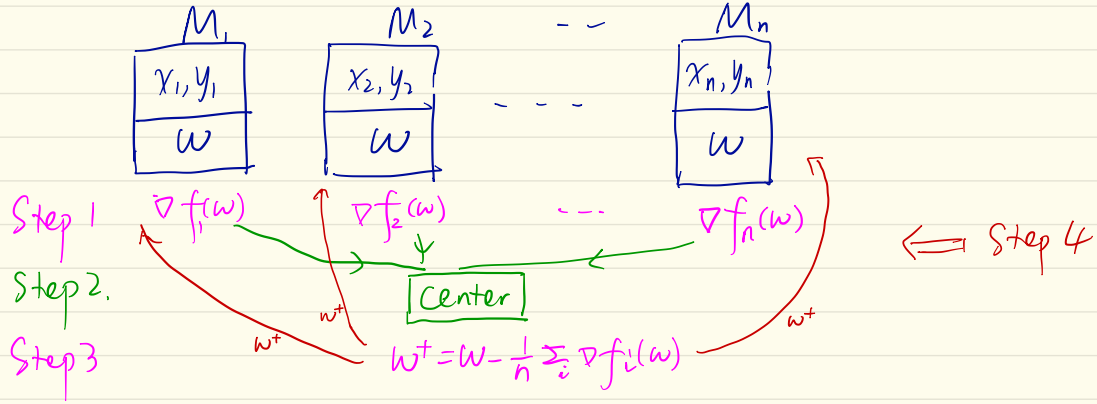
one round: pass $\nabla f_i(w)$

one round: pass w^+

(see more details of the process in next slide)

Ideal Parallel GD

Assume n machines, each machine stores one sample (x_i, y_i), for simplicity of explanation.



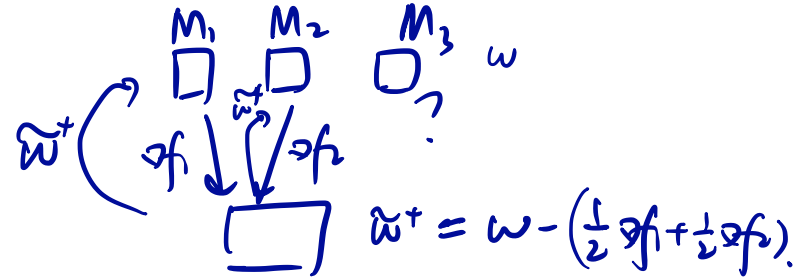
Step 1 Compute partial gradient $\nabla f_i(w)$

Step 2 Communicate $\nabla f_i(w)$ to center

Step 3. Average gradients, i.e., compute $\nabla F(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w)$.
Update $w^+ = w - \nabla F(w)$

Step 4: Communicate w^+ to each machine, for next round of computation.
Go to Step 1.

Comparing SGD and GD (cont'd)



- Compare $\sum_i T_i$ $+C$ and $\max\{T_i\}$ $+2C$?
- Not fair. SGD is "faster": # of epochs for SGD is fewer than GD
  - Assume SGD takes 1 epoch, GD $\sqrt{n}$ epochs (difference in $[1, n]$) to achieve same error, say, ε
- SGD $\sum_i T_i$ $+C$ v.s. $\sqrt{n}(\max\{T_i\} +2C)$

Computation: $\sum_i T_i$ v.s. $\sqrt{n}\max\{T_i\}$

Communication: C v.s. $2\sqrt{n}C$.

- Which one larger? Depends on C, T_i, n

Assume $T_i = T$. nT v.s. $\sqrt{n}T$. (SGD vs GD)

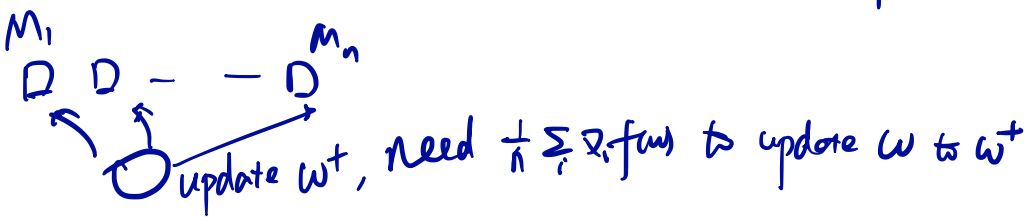
HOGWILD!

- **Bottleneck 1**: Communication cost C can be high.
- **Bottleneck 2**: $\max\{T_i\}$ means waiting for slowest machine. Synchronization issue

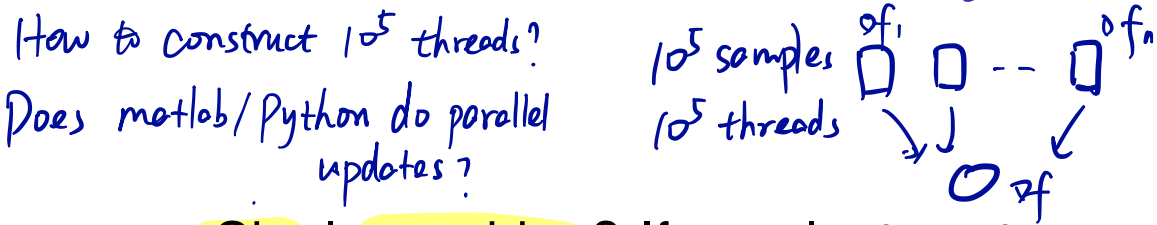
Special case, $\{T_i\} = \{1, 1, 1, \cdots, 1, \sqrt{n}\}$.

$\sum_i T_i \approx n + \sqrt{n}$, $\sqrt{n}\max\{T_i\} \approx n$.

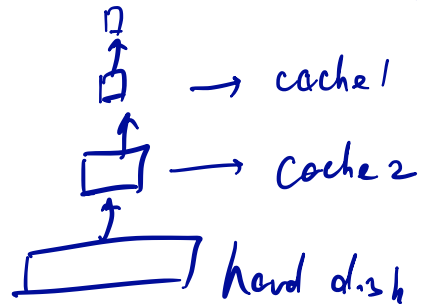
SGD similar to GD.



System Architecture



- Single machine? If your laptop stores all data (say, $500G$), maybe GD could be faster than SGD?
  - Is your laptop/software really doing fully parallel computation?
  - Working memory (cache) is not your hard disk capacity
  - Multi-level memory system?
  - Even random sampling takes time...



(sometimes impossible)

- There is no ideal "single machine"
  - GPU v.s. CPU

- System architecture matters. Beyond the scope of this course...

Final Remark: Finite Sum v.s. Expectation

We only consider finite sum $\min_w \sum_i f_i(w)$ → for one sample x_i

SGD is also used to solve minimization of expectation

$$\min_w E_\xi F(w;\xi),$$

where ξ is a random variable. sample ξ_i, $\nabla F(w;\xi_i)$.

This is another story. We skip the discussion in this class.

- Want to know a bit more about their relations/differences? You can check IE598 post on SGD, or read [Bot16] (see the last few slides)

Outline

Summary

In this lecture, we learned the following:

- Different variants of SGD
- Application of SGD to finite sum
- Convergence: diminishing stepsize + Lipchitz + bounded variance
- Convergence speed: typically faster than GD, since bigger stepsize allowed (in early stage)
  - Though in theory the dependence on ϵ is worse

Summary

In this lecture, we learned the following:

- Different variants of SGD
- Application of SGD to finite sum trick in SGD for $\sum_{i,j}(M_{ij} - x_i y_j)^2$.
- Convergence: diminishing stepsize + Lipchitz + bounded variance $\sum_t \alpha_t = \infty, \sum_t \alpha_t^2 < \infty$
- Convergence speed: typically faster than GD, since bigger stepsize allowed (in early stage)
  - Though in theory the dependence on ϵ is worse
  - Parallel / distributed computation.

References and Further Reading

[BT2000] Bertsekas, Tsitsiklis 2000, "**Gradient Convergence in Gradient methods with Errors**". (easiest to read)

- General framework for incremental gradient methods
- Proposition 1 gives a general convergence result for inexact GD
- Proposition 2 is a corollary of Prop 1, applying to incremental gradient method. This is Proposition 8.1 in this lecture.

[Luo91] Luo, Z. Q., 1991. "**On the Convergence of the LMS Algorithm with Adaptive Learning Rate for Linear Feedforward Networks**," Neural Computation, Vol. 3, pp. 226-245.

- One of the first rigorous analysis of SGD is given in Kohonen 1974; but for constant stepsize, so error doesn't go to zero
- Diminishing stepsize for SGD was proposed in Kushner and Clark 1978
- Luo-91 rigorously proved the convergence of incremental gradient method for the diminishing stepsize rule (no statistical assumption)

Bertsekas,1999, "Nonlinear Programming". Proposition 1.5.1.

- For strongly convex quadratic case, only need $\sum_t \alpha_t = \infty$ to ensure iterates converge

References and Further Reading

[Ber10] Bertsekas 2010, "**Incremental Gradient, Subgradient, and Proximal Methods for Convex Optimization: A Survey**".

- Comprehensive survey for convex case. Cover both cyclic and randomized order
- However, does not cover non-convex case
- A notable fact: for convex function values to converge, only need $\sum_t \alpha_t = \infty$, no need of $\sum_t \alpha_t^2 < \infty$; but need $\sum_t \alpha_t^2 < \infty$ to ensure iterates to converge (Prop 3.4 and Prop 4.3)

[Bot16] Bottou, Curtis, Nocedal 2016, "**Optimization Methods for Large-Scale Machine Learning**", (machine learning perspective of SGD)

- Only cover randomized order
- Cover convergence in nonconvex case (Theorem 4.9, 4.10)
- Cover convergence rate in strongly/non-strongly convex case (Theorem 4.6, 4.7, 4.8)
- Cover both constant and diminishing stepsize

Further Reading on SGD for Neural-nets

[Bot12] Bottou 2012, **Stochastic Gradient Descent Tricks**.

- Discuss a few tricks for applying SGD to machine learning
- including the diminishing stepsize rules

[Le98] LeCun, Bottou, Orr, Muller 1998, **Efficient Backprop**.

- 44 pages. Lots of empirical suggestions
- Most of them are based on some theoretical insight
- Many insights come from eigenvalue analysis (similar to what we've done in class)

Further reading on distributed SGD

Some keywords to search:

HOGWILD!

Asynchronous SGD.

- e.g. Jeff Dean et al. “large scale distributed deep networks” (famous google paper)

Asynchronous coordinate descent.

- e.g. Liu et al. “An Asynchronous Parallel Stochastic Coordinate Descent Algorithm.”

Parameter server.

- e.g. Li et al. 2014 “Scaling Distributed Machine Learning with the Parameter Server”