Numerical Optimization for Machine Learning

Coordinate Optimization and Stochastic Gradient Descent

Mark Schmidt

University of British Columbia

Summer 2022

Last Time: Faster Algorithms than Gradient Descent

- The heavy-ball method

$$w^{k+1} = w^k - \alpha_k \nabla f(w^k) + \beta_k (w^k - w^{k-1}),$$

  is faster on strongly-convex quadratics.
  - And optimizing over α_k and β_k at each step yields conjugate gradient.
- Nesterov's accelerated gradient method,

$$w^{k+1} = v^k - \alpha_k \nabla f(v^k), \quad v^{k+1} = w^{k+1} + \beta_k (w^{k+1} - w^k),$$

  is faster on convex and strongly-convex functions.
  - Restarting schemes have been proposed that adapt to strong-convexity level.
- Newton's method uses second-derivative information (or Hessian approximation),

$$w^{k+1} = w^k - [\nabla^2 f(w^k)]^{-1} \nabla f(w^k),$$

  to achieve local superlinear convergence.
  - Convergence requires line-search, trust-region, or cubic regularization.
- Today: algorithms with lower iteration costs than gradient descent.

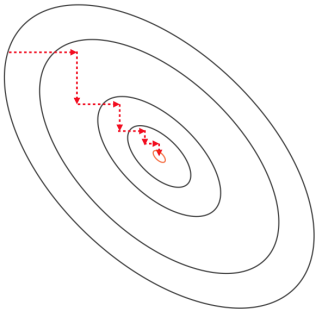
Beyond Gradient Descent

- For high-dimensional problems we often prefer gradient descent over Newton.
  - Gradient descent often requires far more iterations.
  - But iteration cost is only linear in d.
- For very large datasets, even gradient descent iterations can be too slow.
  - If iteration cost is $O(nd)$, we may only be able to do a small number of iterations.
- Two common strategies for yielding even cheaper iterations:
  - Coordinate optimization.
  - Stochastic gradient descent.

Outline

Coordinate Optimization

- Each iteration of coordinate optimization only updates on variable:



- For example, on iteration k we select a variable j_k and set

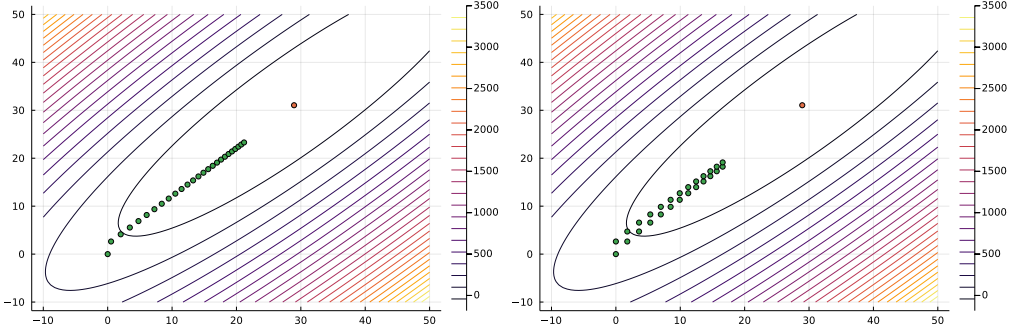
$$w_{j_k}^{k+1} = w_{j_k}^k - \alpha_k \nabla_{j_k} f(w^k),$$

a gradient descent step for one coordinate j_k (other w_j stay the same).

  - This variation is called coordinate descent (many variations exist).

Gradient and Coordinate Descent and Quadratics

- Gradient descent vs. coordinate descent on a quadratic ($\alpha_k = 1/L$):



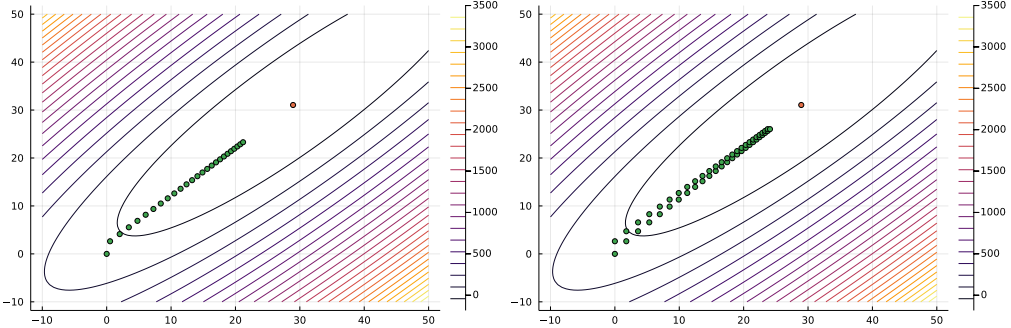
- Where coordinate descent alternates between $j_k = 1$ and $j_k = 2$.
  - Both methods decrease the function on each step.
  - But coordinate descent only updates one coordinate on each step.

Why use Coordinate Descent?

- Theoretically, coordinate descent is a provably bad algorithm:
  - The convergence rate is slower than gradient descent.
  - The iteration cost can be similar to gradient descent.
    - Computing 1 partial derivative may have same cost as computing gradient.
- But it is widely-used in practice:
  - Nothing works better for certain problems.
  - Certain fields think it is the "ultimate" algorithm.
- ??
- Renewed theoretical interest began with a paper by Nesterov in 2010:
  - Showed global convergence rate for randomized coordinate selection.
  - Coordinate descent is faster than gradient descent if iterations are d times cheaper.
    - Sometimes called coordinate-friendly structures.

Gradient and Coordinate Descent and Quadratics

- Gradient descent vs. d coordinate descent steps on a quadratic ($\alpha_k = 1/L$):



- Quadratics have a coordinate-friendly structure.
  - We will see that coordinate descent allows bigger step sizes than gradient descent.

Separable Functions

- For what functions is coordinate descent d times faster than gradient descent?
- The simplest example is separable functions,

$$f(w) = \sum_{j=1}^{d} f_j(w_j),$$

- Here f is the sum of an f_j applied to each w_j, like $f(w) = \frac{\lambda}{2}\|w\|^2 = \sum_{j=1}^{d} \underbrace{\frac{\lambda}{2} w_j^2}_{f_j(w_j)}$.
- Cost of gradient descent vs. coordinate descent:
  - Gradient descent costs $O(d)$ to compute each $f_j'(w_j^k)$.
  - Coordinate descent costs $O(1)$ to compute the *one* $f_{j_k}'(w_{j_k}^k)$.
- In fact, for separable functions you should only use coordinate optimization.
  - The variables w_j have "separate" effects, so can be minimized independently.

Pairwise-Separable Functions

- A more interesting example is pairwise-separable functions,

$$f(w) = \sum_{i=1}^{d} \sum_{j=1}^{d} f_{ij}(w_i, w_j),$$

  which depend on a function of each pair of variables.
  - This includes quadratic functions.

- An example is label propagation for semi-supervised learning.
  - In this application, each f_{ij} measures how similar labels are between neighbours.

- Cost of gradient descent vs. coordinate descent:
  - Double-sum has $O(d^2)$ terms.
    - Gradient descent needs to compute gradient of all these terms.
  - Each w_j only appears in $O(d)$ terms.
    - Coordinate optimization only needs to use these terms.

Label Propagation

- The label propagation example looks a bit more like this:

$$f(w) = \sum_{j=1}^{d} f_j(w_j) + \sum_{(i,j)\in E} f_{ij}(w_i, w_j),$$

  where E is a set of (i, j) pairs ("edges" in a graph).

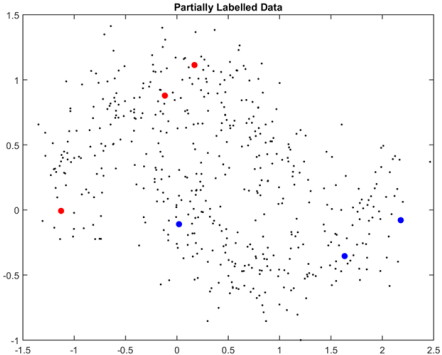
- Adding a separable function doesn't change costs.
  - We could just combine each f_j with one f_{ij}.

- Restricting (i, j) to E makes gradient descent cheaper:
  - Now costs $O(|E|)$ to compute gradient.
  - Coordinate descent could also cost $O(|E|)$ if degree of j_k is $O(|E|)$.

- Coordinate descent is still d times faster in expectation if you randomly pick j_k.
  - Each f'_{ij} is needed with probability $2/d$.
  - So expected cost of $O(|E|/d)$ to compute one partial derivative of f.

Label Propagation with Coordinate Optimization

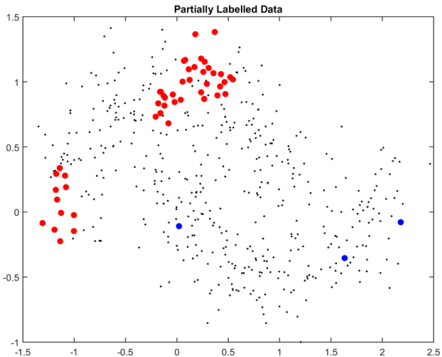
- Label propagation with coordinate optimization (rounding to nearest label):



- Starts with a small number of labeled examples.
  - Optimizing objective "propagates" labels to unlabeled examples.

Label Propagation with Coordinate Optimization

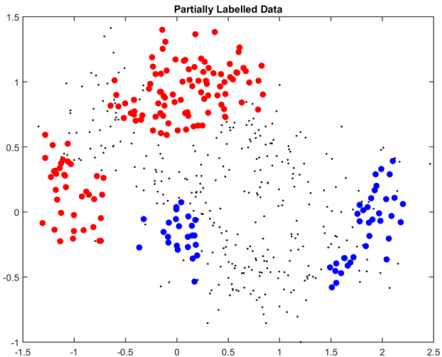
- Label propagation with coordinate optimization (rounding to nearest label):



- Starts with a small number of labeled examples.
  - Optimizing objective "propagates" labels to unlabeled examples.

Label Propagation with Coordinate Optimization

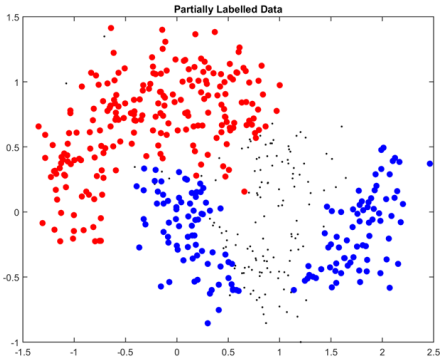
- Label propagation with coordinate optimization (rounding to nearest label):



- Starts with a small number of labeled examples.
  - Optimizing objective "propagates" labels to unlabeled examples.

Label Propagation with Coordinate Optimization

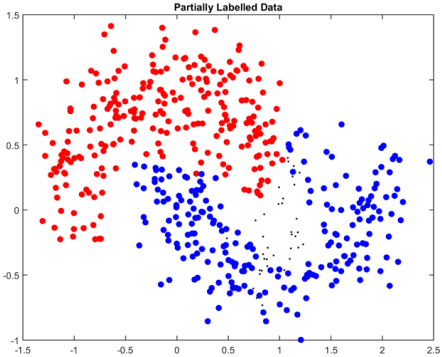
- Label propagation with coordinate optimization (rounding to nearest label):



- Starts with a small number of labeled examples.
  - Optimizing objective "propagates" labels to unlabeled examples.

Label Propagation with Coordinate Optimization

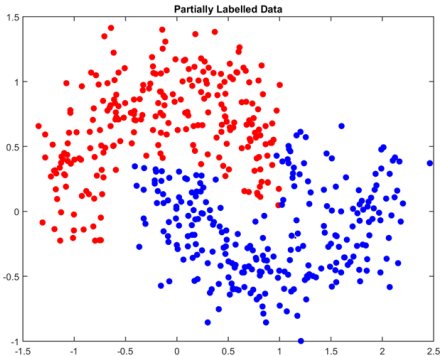
- Label propagation with coordinate optimization (rounding to nearest label):



- Starts with a small number of labeled examples.
  - Optimizing objective "propagates" labels to unlabeled examples.

Label Propagation with Coordinate Optimization

- Label propagation with coordinate optimization (rounding to nearest label):



- Starts with a small number of labeled examples.
  - Optimizing objective "propagates" labels to unlabeled examples.

Linear Compositions

- Another coordinate-friendly structure is linear compositions,

$$f(w) = g(Aw),$$

for a matrix an $n \times d$ matrix X and smooth function g.

  - Includes least squares and logistic regression.

- It is still coordinate friendly if we add a separable function,

$$f(w) = g(Aw) + \sum_{j=1}^{d} f_j(w_j),$$

like an L2-regularizer.

- Key idea: you can track Aw^k as you go for a cost $O(n)$ instead of $O(nd)$.

Efficiently Tracking Aw^k for Linear Compositions

- For linear compositions problems,

$$f(w) = g(Aw),$$

the partial derivatives on iteration k has the form.

$$\nabla_j f(w^k) = a_j^\top g'(Aw^k),$$

where a_j is column j of A.

- If we have Aw^k, this costs $O(n)$ instead of $O(nd)$ for the full gradient.

(Assuming g' costs $O(n)$)

- We can track the product Aw^k as we go with $O(n)$ cost,

$$Aw^{k+1} = A(w^k + \gamma_k e_{j_k}) = \underbrace{Aw^k}_{\text{old value}} + \gamma_k \underbrace{Xe_{j_k}}_{O(n)},$$

this allows computing partial derivatives and implement line-search steps in $O(n)$.

Other Coordinate-Friendly Structures

- Other problems with coordinate-friendly strucutres:
  - Matrix factorization (and tensor factorization) problems like PCA (covered 340),

$$f(Z, W) = \frac{1}{2}\|ZW - X\|^2.$$

  - Log-determinant problems like fitting Gaussians (covered in 440),

$$f(\Theta) = \mathsf{Tr}(S\Theta) - \log|\Theta|.$$

  - Convex extensions of sub-modular functions.

- On the other hand, neural networks are usually not coordinate friendly.
  - Would need something like "number of units after first hidden layer is tiny".

Outline

Analyzing Coordinate Descent

- To analyze coordinate descent, we can write it as

$$w^{k+1} = w^k - \alpha_k \nabla_{j_k} f(w^k) e_{j_k},$$

where "elementary vector" e_j has a zero in every position except j,

$$e_3^\top = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

- We usually assume that each $\nabla_j f$ is L-Lipshitz ("coordinate-wise Lipschitz"),

$$|\nabla_j f(w + \gamma e_j) - \nabla_j f(w)| \leq L|\gamma|,$$

which for $\mathcal{C}^2$ functions is equivalent to $|\nabla^2_{jj} f(w)| \leq L$ for all j.

(diagonals of Hessian are at most L)

- This is not a stronger assumption than for gradient descent:
  - If the gradient is L-Lipschitz then it is also coordinate-wise L-Lipschitz.

Convergence Rate of Coordinate Optimization

- Coordinate-wise Lipschitz assumption implies descent lemma coordinate-wise,

$$f(w^{k+1}) \leq f(w^k) + \nabla_j f(w^k)(w^{k+1} - w^k)_j + \frac{L}{2}(w^{k+1} - w^k)_j^2,$$

for any w^{k+1} and w^k that only differ in coordinate j.

- With $\alpha_k = 1/L$ (for simplicity), plugging in $(w^{k+1} - w^k) = -(1/L)e_{j_k}\nabla_{j_k} f(w^k)$ gives

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}|\nabla_{j_k} f(w^k)|^2,$$

a progress bound based on only updating coordinate j_k.

Convergence Rate of Randomized Coordinate Optimization

- Our bound for updating coordinate j_k is

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}|\nabla_{j_k} f(w^k)|^2,$$

  so progress depends on which j_k that we choose.

- Let's consider expected progress with random selection of j_k,

$$\begin{aligned}
\mathbb{E}[f(w^{k+1})] &\leq \mathbb{E}\left[f(w^k) - \frac{1}{2L}|\nabla_{j_k} f(w^k)|^2\right] && \text{(expectation wrt } j_k \text{ given } w^k\text{)} \\
&= \mathbb{E}[f(w^k)] - \frac{1}{2L}\mathbb{E}[|\nabla_{j_k} f(w^k)|^2] && \text{(linearity of expectation)} \\
&= \underbrace{f(w^k)}_{\text{no } j_k} - \frac{1}{2L}\sum_{j=1}^{d} p(j_k = j)|\nabla_j f(w^k)|^2 && \text{(definition of expectation)}
\end{aligned}$$

- Above, expectation is conditioned on all iterates/gradients/step-sizes up to time k.

Convergence Rate of Randomized Coordinate Optimization

- The bound from the previous slide is

$$E[f(w^{k+1})] \leq f(w^k) - \frac{1}{2L}\sum_{j=1}^{d} p(j_k = j)|\nabla_j f(w^k)|^2.$$

- Let's choose j_k uniformly at random in this bound, $p(j_k = j) = 1/d$.

$$\begin{aligned}\mathbb{E}[f(w^{k+1})] &\leq f(w^k) - \frac{1}{2L}\sum_{j=1}^{d}\frac{1}{d}|\nabla_j f(w^k)|^2 \\ &= f(w^k) - \frac{1}{2dL}\sum_{j=1}^{d}|\nabla_j f(w^k)|^2 \\ &= f(w^k) - \frac{1}{2dL}\|\nabla f(w^k)\|^2.\end{aligned}$$

Convergence Rate of Randomized Coordinate Optimization

- Our guaranteed progress bound for randomized coordinate optimization,

$$\mathbb{E}[f(w^{k+1}))] \leq f(w^k) - \frac{1}{2dL}\|\nabla f(w^k)\|^2.$$

- If we use strongly convexity or PL and recurse carefully (see bonus) we get

$$\mathbb{E}[f(w^k)] - f^* \leq \left(1 - \frac{\mu}{dL}\right)^k [f(w^0) - f^*].$$

- If we want $O(\left(1 - \frac{\mu}{dL}\right)^k) \leq \epsilon$, we need $O\left(d\frac{L}{\mu}\log(1/\epsilon)\right)$ iterations.

- For PL functions gradient descent needs $O\left(\frac{L}{\mu}\log(1/\epsilon)\right)$ iterations.

- So coordinate optimization needs d-times as many iterations?

Randomized Coordinate Optimization vs. Gradient Descent

- If coordinate descent step are d-times cheaper then both algorithms need

$$O\left(\frac{L}{\mu}\log(1/\epsilon)\right),$$

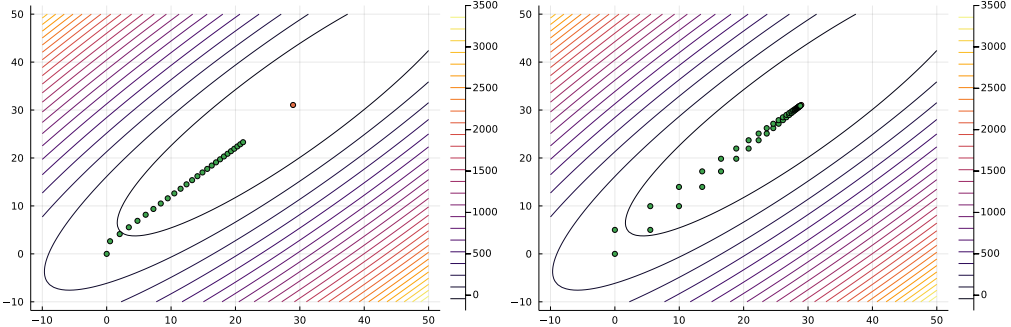
  in terms of "gradient descent iteration cost".
- So why prefer coordinate optimization?

- The Lipschitz constants L are different.
  - Let L_f be maximum gradient changes if you change *all* coordinates.
  - Let L_c be the maximum partial derivative changes if you change *one* coordinate.
  - Gradient descent uses L_f and coordinate optimization uses L_c.

- Since $L_c \leq L_f$, coordinate optimization is faster.
  - The gain is because coordinate descent allows bigger step-sizes.
  - For [non-]convex functions, similar trade-off: $O(L_f/\epsilon)$ vs. $O(dL_c/\epsilon)$ iterations.
    - Comparison is harder if we start adding practical tricks like line-search.

Gradient and Coordinate Descent and Quadratics

- Gradient descent ($\alpha = 1/L_f$) vs. d coordinate descent steps ($\alpha = 1/L_c$):



- Coordinate descent allows larger step sizes than gradient descent.
  - In this case $L_f = 3.8$ and $L_c = 2$.

Setting the Step Size in Practice

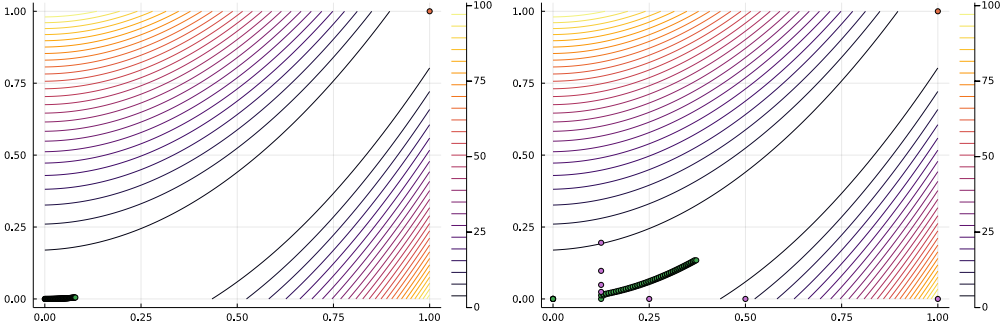
- As with gradient descent, you can often take bigger steps than $\alpha_k = 1/L$.
  - Though the difference is typically not as dramatic.

- Standard choices include:
  - Approximating L using the backtracking procedure we have previously discussed.
  - Approximating an L_j for each coordinate (works much better).
  - Armijo backtracking.
  - Exact line-search (for quadratics, or numerically for other functions).
  - Coordinate-wise Newton steps (to initialize Armijo or implement line-search).
    - This is cheap because we are only working with coordinate.

- Coordinate-friendly structures often allow efficient line-search and Newton steps.
  - For example, computing Hessian diagonal has same cost as partial derivative.

Practical Step Sizes for Coordinate Descent

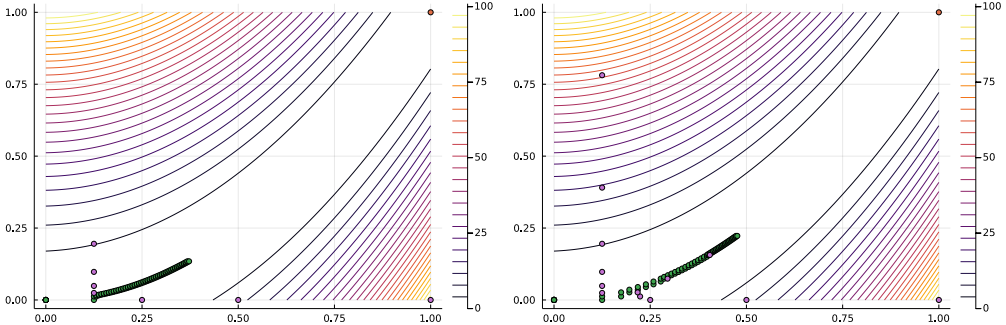
- Coordinate descent with $\alpha_k = 1/L_c$ and $\alpha_k = 1/\hat{L}_c$ on Rosenbrock:



- As before, estimating the Lipschitz constant works better than knowing/using it.
  - $L_c = 1202$ on $[0,1]^2$ while final $\hat{L}_c = 256$ for this function.

Practical Step Sizes for Coordinate Descent

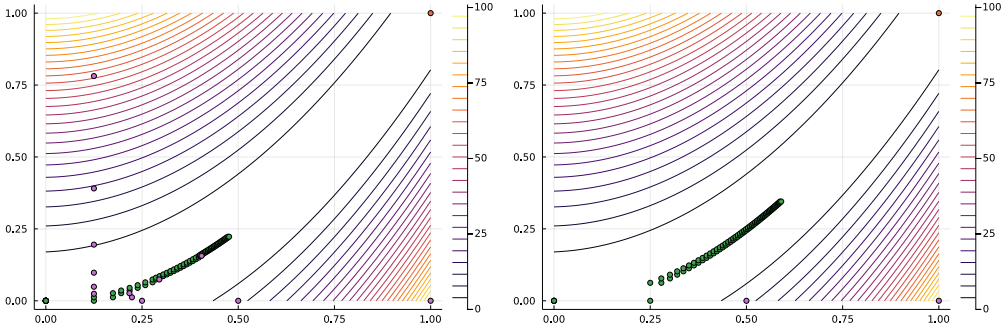
- Coordinate descent with $\alpha_k = 1/\hat{L}_c$ and $\alpha_k = 1/\hat{L}_j$ on Rosenbrock:



- Using a Lipshitz constant for each coordinate (rigtht) works better.
  - For this problem $L_1 = 1202$ and $L_2 = 200$ (final estimate was 256 for both).

Practical Step Sizes for Coordinate Descent

- Coordinate descent with $\alpha_k = 1/\hat{L}_j$ and Armijo initialized with Newton:



- Newton initialization allows step size to increase across iterations.
  - Step size $\alpha_k = 1$ (pure Newton) was only rejected by Armijo on second iteration.
  - But this "coordinate Newton" method does not have superlinear convergence.

Outline

Cyclic and Random-Shuffling Coordinate Selection

- An alternative to random selection of j_k is cyclic selection:
  1. Set $j_1 = 1$, $j_2 = 2$, $\ldots$, $j_d = d$.
  2. Set $j_{d+1} = 1$, $j_{d+2} = 2$, $\ldots$, $j_{2d} = d$.
  3. Set $j_{2d+1} = 1$, $j_{2d+2} = 2$, $\ldots$, $j_{3d} = d$.
- Cyclic often outperforms random in practice, but is worse in theory.
  - For some problems, a bad ordering leads to provably-bad performance for cyclic.

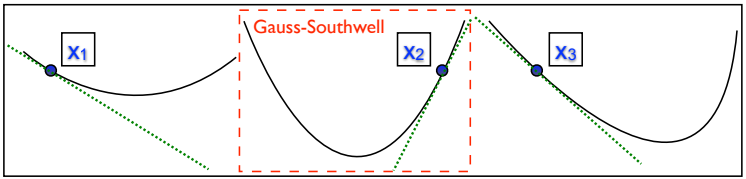
- A hybrid between cyclic and random is using random shuffling:
  1. Chooses random permutation r and sets $j_1 = r[1]$, $j_2 = r[2]$, $\ldots$, $j_d = r[d]$.
  2. Choosesrandom permutation r and sets $j_{d+1} = r[1]$, $j_{d+2} = r[2]$, $\ldots$, $j_{2d} = r[d]$.
  3. Chooses random permutation r and sets $j_{2d+1} = r[1]$, $j_{2d+2} = r[2]$, $\ldots$, $j_{3d} = r[d]$.
- Recent work shows that this fixes cyclic coordinate descent in some settings.
  - Conjectured that random shuffling is faster than cyclic and random.

Gauss-Southwell: Greedy Coordinate Descent

- Instead of cyclic or random, there are also greedy coordinate selection methods.
- The classic greedy method is the Gauss-Southwell rule,

$$j_k \in \underset{j}{\text{argmax}}\{|\nabla_j f(w^k)|\},$$

which choose the coordinate with the largest directional derivative.



Implementing Greedy Coordinate Descent

- Is it possible to be coordinate-friendly with respect to the Gauss-Southwell rule?
  - Can we ever find largest $|\nabla_j f(w)|$ value d-times cheaper than computing $\nabla f(w)$?

- Yes!
  - Possible for pairwise-separable functions if maximum degree $= O($average degree$)$.
    - Includes lattice-structured graphs, complete graphs, and Facebook graph.
  - This is possible for linear compositions with a row-sparse and column-sparse matrix.
    - Requires (max non-zeroes in each row and column)$^2 = O($(total non-zeroes)/d$)$.

- Key idea: track the gradient values and track the max with a max-heap.
  - The above restrictions guarantee that the structure is coordinate friendly.
    - Up to a log factor due to the heap operations.

Analyzing Greedy Coordinate Descent

- Our bound on the progress if we choose coordinate j_k is

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}|\nabla_{j_k} f(w^k)|^2.$$

and the Gauss-Southwell rule chooses

$$j_k \in \underset{j}{\text{argmax}}\{|\nabla_j f(w^k)|\}.$$

- This leads to a progress bound of

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}\|\nabla f(w^k)\|_\infty^2,$$

which is similar to gradient descent but in a different norm.

  - Unlike random coordinate descent, this is dimension independent (no d).

Gauss-Southwell Convergence Rate

- The progress bound under the greedy Gauss-Southwell rule is

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}\|\nabla f(w^k)\|_\infty^2,$$

and for PL functions this leads to a rate of

$$f(w^k) - f^* \leq \left(1 - \frac{\mu_1}{L}\right)^k [f(w^0) - f^*],$$

where μ_1 is the PL constant in the ∞-norm

$$\mu_1[f(w) - f^*] \leq \frac{1}{2}\|f(w)\|_\infty^2.$$

- This is faster than random because $\frac{\mu}{d} \leq \mu_1 \leq \mu$ (by norm equivalences).
  - The μ_1-PL condition is implied by strong-convexity in the 1-norm.

Individual Lipschitz Constants and Non-Uniform Sampling

- For randomized selection, is uniform sampling distribution?
- You can go faster if you have an L_j for each coordinate:

$$|\nabla_j f(w + \gamma e_j) - \nabla_j f(w)| \leq L_j |\gamma|.$$

  - If $f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$, we would have $L_j = \|x_j\|^2 + \lambda$.
    - Where x_j is column j of X.

- Consider sampling j_k proportional to these Lipschitz constants,

$$p(j_k = j) = \frac{L_j}{\sum_{j'=1}^{d} L_{j'}},$$

  - "Sample more often the coordinate where the gradient can change quickly".

Progress Bound under Lipschitz Sampling

- Our bound for updating coordinate j_k with a step size of $\alpha_k = 1/L_{j_k}$ is

$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L_{j_k}} |\nabla_{j_k} f(w^k)|^2,$$

- Taking expectation with Lipschitz sampling gives

$$\mathbb{E}[f(w^{k+1}] \leq f(w^k) - \frac{1}{2} \sum_{j=1}^{d} \frac{p(j_k = j)}{L_j} |\nabla_j f(w^k)|^2$$

$$\mathbb{E}[f(w^{k+1}] \leq f(w^k) - \frac{1}{2} \sum_{j=1}^{d} \frac{L_j}{L_j \sum_{j'=1}^{d} L_{j'}} |\nabla_j f(w^k)|^2$$

$$\mathbb{E}[f(w^{k+1}] \leq f(w^k) - \frac{1}{2 \sum_{j'=1}^{d} L_{j'}} \|\nabla f(w^k)\|^2.$$

- where compared to uniform result replaces the factor dL_c with smaller $\sum_{j'=1}^{d} L_{j'}$.

Convergence Rate under Lipschitz Sampling

- We can re-write the progress from the previous slide as

$$\mathbb{E}[f(w^{k+1}] \leq f(w^k) - \frac{1}{2d\bar{L}}\|\nabla f(w^k)\|^2.$$

  where $\bar{L}$ is the average Lipschitz constant.
  - Our uniform sampling result depends on the maximum L_j.

- If we assume PL then we obtain a rate for Lipschitz sampling of

$$\mathbb{E}[f(w^k)] - f^* \leq \left(1 - \frac{\mu}{d\bar{L}}\right)^k [f(w^0) - f^*],$$

- It is also possible to analyze uniform sampling under these larger step sizes.
  - If you do this, Lipschitz sampling is not necessarily faster.

Lipschitz Sampling - Estimating $\hat{L}_j$

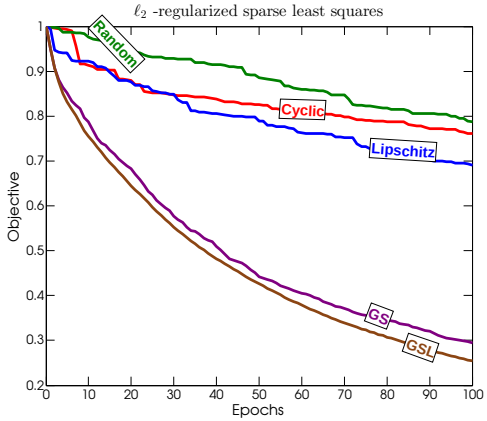
- Instead of using each L_j, we could approximate them with backtracking.
- But sampling from the $\hat{L}_j$ can go badly:
  - Start with all $L_j = 1$: $\begin{bmatrix}1 & 1 & 1 & 1 & 1\end{bmatrix}$.
  - Selected coordinate 4 and backtrack, giving $\begin{bmatrix}1 & 1 & 1 & 1024 & 1\end{bmatrix}$.
  - Sample coordinate 4 a hundred times before trying a different coordinate.
- My trick to avoid this problem:
  - Let m be the number of coordinates that chosen at least once ("visited").
  - With probability m/d, sample from "visited" coordinates proportional to their $\hat{L}_j$.
  - With probability $1 - m/d$, sample uniformly from the "unvisited" coordinates.
    - Guarantees you solve the "coupon collector" problem at same speed as uniform.
    - After $O(d \log d)$ iterations, all coordinates have an initialized estimate of $\hat{L}_j$.

Lipschitz Sampling - Cost of Sampling

- Cost of sampling from a non-uniform distribution is $O(d)$.
  - Which may not be coordinate-friendly for some problems.
- But cost sampling k variables is only $O(d + k \log d)$.
  - $O(d)$ once to compute cumunlative distribution function.
  - $O(\log d)$ to generate each sample by binary search (see CPSC 440).
- You can maintain the above cost if one L_j changes on each iteration.
  - By maintaining the L_j in a sum-heap.
- But Lipschitz sampling may still not be coordinate friendly even when uniform is.
  - Requires similar conditions to Gauss-Southwell to be coordinate friendly.

Numerical Comparison of Coordinate Selection Rules

- Comparison on problems where greedy and random have similar cost:



- Greedy rules tend to work a lot better when they are coordinate friendly.

Gauss-Southwell-Lipschitz

- Our bound on the progress with an L_j for each coordinate is

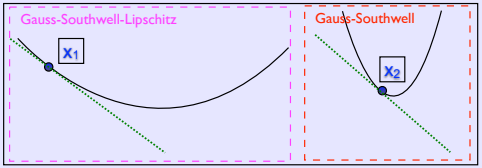
$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L_{j_k}}|\nabla_{j_k} f(w^k)|^2.$$

- The best coordinate to update according to this bound is

$$j_k \in \underset{j}{\text{argmax}} \frac{|\nabla_j f(w^k)|^2}{L_j}$$

which is called the Gauss-Southwell-Lipschitz rule.

  - "If gradients are similar, pick the one that changes more slowly".



- This is the optimal update for quadratic functions.

Block Coordinate Descent

- Instead of updating 1 variable, block coordinate descent updates a "block".

- Examples where you might want to do this:
  - Coordinate descent steps converge too slow or don't fully-utilize parallel resources:
    - Better to do a Newton step on 50 variables on each iteration?
  - Problems with special structure, like multi-class logistic regression on n examples,

$$f(W) = \sum_{i=1}^{n} \left[-w_{y^i}^\top x^i + \log\left(\sum_c \exp\left(w_c^\top x^i \right) \right) \right],$$

    the cost of computing/updating 1 partial derivative w_c^j is the same as for w_c.
    - So you could update an entire vector for cost of updating 1 parameter.

- There also exist accelerated coordinate descent methods.
  - Form of update is similar to Nesterov's accelerated gradient method.
  - Requires careful implementation for momentum to be coordinate friendly.

Outline

Finite-Sum Optimization Problems

- We have been discussing optimizing a generic function f,

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(w).$$

- But in machine learning f is often an average over functions f_i,

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} f_i(w),$$

  where typically f_i will measure the error on training example i.
  - For least squares we would have $f_i(w) = \frac{1}{2}(w^T x_i - y^i)^2$.

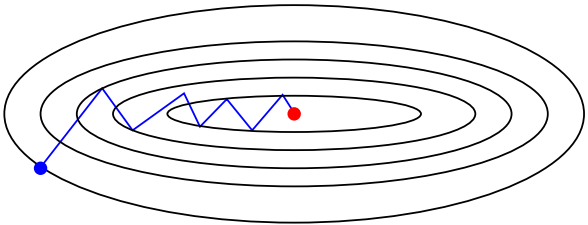
- Gradient methods are effective when d is very large.
- What if number of training examples n is very large?
  - E.g., ImageNet has ≈ 14 million annotated images.

Stochastic vs. Deterministic Gradient Methods

- We consider minimizing $f(w) = \frac{1}{n}\sum_{i=1}^{n} f_i(w)$.
- Deterministic gradient method [Cauchy, 1847]:

$$w^{k+1} = w^k - \alpha_k \nabla f(w^k) = w^k - \frac{\alpha_k}{n}\sum_{i=1}^{n} \nabla f_i(w^k).$$

  - Iteration cost is linear in n.
  - Convergence with constant α_k or line-search.



Stochastic vs. Deterministic Gradient Methods

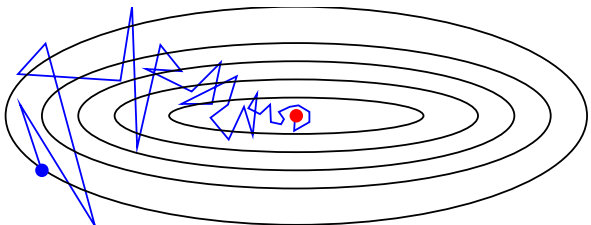
- Stochastic gradient method [Robbins & Monro, 1951]:
  - Random selection of i_k from $\{1, 2, \ldots, n\}$.

$$w^{k+1} = w^k - \alpha_k \nabla f_{i_k}(w^k).$$

  - With $p(i_k = i) = 1/n$, the stochastic gradient is an unbiased estimate of gradient,

$$\mathbb{E}[\nabla f_{i_k}(w)] = \sum_{i=1}^{n} p(i_k = i) \nabla f_i(w) = \sum_{i=1}^{n} \frac{1}{n} \nabla f_i(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w) = \nabla f(w).$$

  - Iteration cost is independent of n.
  - Convergence requires $\alpha_k \to 0$.



Stochastic vs. Deterministic Gradient Methods

Stochastic iterations are n times faster, but how many iterations are needed?

- If ∇f is Lipschitz continuous then we have:

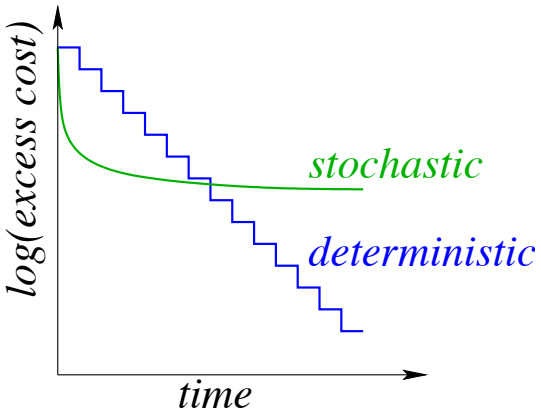
Assumption	Deterministic Gradient Descent	Stochastic Gradient Descnet
PL	$O(\log(1/\epsilon))$	$O(1/\epsilon)$
Convex	$O(1/\epsilon)$	$O(1/\epsilon^2)$
Non-Convex	$O(1/\epsilon)$	$O(1/\epsilon^2)$

- Stochastic has low iteration cost but slow convergence rate.
  - Sublinear rate even in strongly-convex case.
  - Bounds are unimprovable under standard assumptions.
    - Oracle returns an unbiased gradient approximation with bounded variance.

- Momentum and Newton-like methods do not improve rates in stochastic case.
  - Can only improve constant factors (bottleneck is variance, not condition number).

Stochastic vs. Deterministic Convergence Rates

Plot of convergence rates in strongly-convex case:



Stochastic will be superior for low-accuracy/time situations.

Progress Bound for Stochastic Gradient Method

- The stochastic gradient descent (SGD) update is

$$w^{k+1} = w^k - \alpha_k \nabla f_{i_k}(w^k).$$

- Recall the the descent lemma applied to w^{k+1} and w^k,

$$f(w^{k+1}) \leq f(w^k) + \nabla f(w^k)^\top (w^{k+1} - w^k) + \frac{L}{2}\|w^{k+1} - w^k\|^2.$$

- Plugging in SGD iteration $(w^{k+1} - w^k) = -\alpha_k \nabla f_{i_k}(w^k)$ gives

$$f(w^{k+1}) \leq f(w^k) - \alpha_k \nabla f(w^k)^\top \nabla f_{i_k}(w^k) + \alpha_k^2 \frac{L}{2}\|\nabla f_{i_k}(w^k)\|^2.$$

Progress Bound for Stochastic Gradient Method

- So far any choice of α_k and i_k we have

$$f(w^{k+1}) \leq f(w^k) - \alpha_k \nabla f(w^k)^\top \nabla f_{i_k}(w^k) + \alpha_k^2 \frac{L}{2} \|\nabla f_{i_k}(w^k)\|^2.$$

- Let's take the expectation and assume α_k does not depend on i_k,

$$\begin{aligned}\mathbb{E}[f(w^{k+1})] &\leq \mathbb{E}[f(w^k) - \alpha_k \nabla f(w^k)^\top \nabla f_{i_k}(w^k) + \alpha_k^2 \frac{L}{2} \|\nabla f_{i_k}(w^k)\|^2] \\ &= f(w^k) - \alpha_k \nabla f(w^k)^\top \mathbb{E}[\nabla f_{i_k}(w^k)] + \alpha_k^2 \frac{L}{2} \mathbb{E}[\|\nabla f_{i_k}(w^k)\|^2],\end{aligned}$$

  where the second line uses linearity of expectation.

- Under uniform sampling $\mathbb{E}[\nabla f_{i_k}(w^k)] = \nabla f(w^k)$ (unbiased) so this gives

$$\mathbb{E}[f(w^{k+1})] \leq f(w^k) - \alpha_k \underbrace{\|\nabla f(w^k)\|^2}_{\text{good}} + \alpha_k^2 \underbrace{\frac{L}{2} \mathbb{E}[\|\nabla f_{i_k}(w^k)\|^2]}_{\text{bad}}.$$

Convergence Rate of Stochastic Gradient Method

- So a progress bound for stochastic gradient is

$$\mathbb{E}[f(w^{k+1})] \leq f(w^k) - \alpha_k \underbrace{\|\nabla f(w^k)\|^2}_{\text{good}} + \alpha_k^2 \underbrace{\frac{L}{2}\mathbb{E}[\|\nabla f_{i_k}(w^k)\|^2]}_{\text{bad}}.$$

- "Good" term looks like usual measure of progress: big gradient $\rightarrow$ big progress.
- "Bad" term is the problem: less progress if gradients are very different.
  - And now choosing $\alpha_k = 1/L$ might not be small enough.
  - But we can control badness: if α_k is small then $\alpha_k >> \alpha_k^2$.

- Step-size α_k controls how fast we move towards solution.
- And squared step-size α_k^2 controls how much noise moves us away.
  - This term destroys linear convergence even under PL.

Bounded Variance Assumption

- We will first analyze SGD assuming only that f is bounded below.
  - So it could be non-convex.
- To bound the effect of the noise, we assume for some σ that

$$\mathbb{E}[\|\nabla f_i(w)\|^2] \leq \sigma^2,$$

  for all w.
- This assumption is strong.
  - It implies gradients are bounded, and cannot hold for PL functions on all of $\mathbb{R}^d$.
- Common weaker assumptions include:
  - Variation of gradients is bounded: $\mathbb{E}[\|\nabla f_i(w) - \nabla f(w)\|^2] \leq \sigma^2$ (for all w).
  - Variation of gradients at solution is bounded: $\mathbb{E}[\|\nabla f_i(w^*)\|^2] \leq \sigma^2$.
- Get similar results under these assumptions, but they are a bit uglier.

Convergence Rate of Stochastic Gradient Method

- Using our noise assumption inside the progress bound,

$$\begin{aligned}\mathbb{E}[f(w^{k+1})] &\leq f(w^k) - \alpha_k \|\nabla f(w^k)\|^2 + \alpha_k^2 \frac{L}{2} \mathbb{E}[\|\nabla f_{i_k}(w^k)\|^2] \\ &\leq f(w^k) - \alpha_k \|\nabla f(w^k)\|^2 + \alpha_k^2 \frac{L\sigma^2}{2}.\end{aligned}$$

- As before, re-arrange to get the gradient norm on the left side,

$$\alpha_k \|\nabla f(w^k)\|^2 \leq f(w^k) - \mathbb{E}[f(w^{k+1})] + \alpha_k^2 \frac{L\sigma^2}{2}.$$

- Sum this up (and use iterated expectation) to get

$$\sum_{k=1}^{t} \alpha_{k-1} \mathbb{E}\|\nabla f(w^{k-1})\|^2 \leq \sum_{k=1}^{t} [\mathbb{E} f(w^{k-1}) - \mathbb{E} f(w^k)] + \sum_{k=1}^{t} \alpha_{k-1}^2 \frac{L\sigma^2}{2}.$$

Convergence Rate of Stochastic Gradient Method

- The bound from the previous slide:

$$\sum_{k=1}^{t} \alpha_{k-1} \underbrace{\mathbb{E}\|\nabla f(w^{k-1})\|^2}_{\text{bound by min}} \leq \sum_{k=1}^{t} \underbrace{[\mathbb{E}f(w^{k-1}) - \mathbb{E}f(w^k)]}_{\text{telescope}} + \sum_{k=1}^{t} \alpha_{k-1}^2 \underbrace{\frac{L\sigma^2}{2}}_{\text{no } k}.$$

- Applying the above operations gives

$$\min_{k=0,1,\ldots,t-1} \{\mathbb{E}\|\nabla f(w^k)\|^2\} \sum_{k=0}^{t-1} \alpha_k \leq f(w^0) - \mathbb{E}f(w^t) + \frac{L\sigma^2}{2} \sum_{k=0}^{t-1} \alpha_k^2.$$

- Using $\mathbb{E}f(w^k) \geq f^*$ and dividing both sides by $\sum_k \alpha_{k-1}$ gives

$$\min_{k=0,1,\ldots,t-1} \{\mathbb{E}\|\nabla f(w^k)\|^2\} \leq \frac{f(w^0) - f^*}{\sum_{k=0}^{t-1} \alpha_k} + \frac{L\sigma^2}{2} \frac{\sum_{k=0}^{t-1} \alpha_k^2}{\sum_{k=0}^{t-1} \alpha_k}.$$

Convergence Rate of Stochastic Gradient Method

- The final bound:

$$\min_{k=0,1,\ldots,t-1}\{\mathbb{E}\|\nabla f(w^k)\|^2\} \leq \frac{f(w^0)-f^*}{\sum_{k=0}^{t-1}\alpha_k} + \frac{L\sigma^2}{2}\frac{\sum_{k=0}^{t-1}\alpha_k^2}{\sum_{k=0}^{t-1}\alpha_k}.$$

- Something weird: we do not observe $\nabla f(w^k)$ so do not know which one is min.
  - Bonus slides show how you can get same rate with a random iterate.

- First term on right above looks like the deterministic bound.
  - If you use $\alpha_k = 1/L$ the first term would be $L(f(w^0)-f^*)/\sum_k \alpha_k$.

- But due to stochasticity, convergence rate is determined by $\sum_k \alpha_k^2 / \sum_k \alpha_k$.
  - We want $\sum_k \alpha_k$ to grow quickly but $\sum_k \alpha_k^2$ to grow slowly.

Convergence Rate of Stochastic Gradient Method

- How does $\sum_k \alpha_k^2 / \sum_k \alpha_k$ behave under different step size sequences?

- Classic decreasing step-sizes: set $\alpha_k = \alpha/k$ for some α.
  - Gives $\sum_k \alpha_k = O(\log(t))$ and $\sum_k \alpha_k^2 = O(1)$, so error at k is $O(1/\log(k))$.

- Bigger decreasing step-sizes: set $\alpha_k = \alpha/\sqrt{k}$ for some α.
  - Gives $\sum_k \alpha_k = O(\sqrt{k})$ and $\sum_k \alpha_k^2 = O(\log(k))$, so error at k is $O(\log(k)/\sqrt{k})$.

- Constant step-sizes: set $\alpha_k = \alpha$ for some α.
  - Gives $\sum_k \alpha_k = k\alpha$ and $\sum_k \alpha_k^2 = k\alpha^2$, so error at k is $O(1/\alpha k) + O(\alpha)$.
  - First term converges like $O(1/k)$ but second term does not converge to 0.

- We typically do not need error of exactly 0 but are happy with some ϵ.
  - For any ϵ, there is a constant step size that achieves this.

Summary

- Coordinate optimization updates one variable at a time.
  - Faster rates than gradient descent for coordinate-friendly structures.
    - We can update d variables for the cost of computing gradient.
- We discussed Lipschitz sampling and Gauss-Southwell rule.
  - Faster convergence than uniform sampling.
  - Require extra assumption to preserve coordinate-friendly property.
- Stochastic gradient descent uses a stochastic approximation of gradient.
  - Slower rates than deterministic gradient descent.
  - Analyzed assuming approximation is unbiased with bounded variance.
  - Requires decreasing step size to converge, but constant may work better.

- Next time: faster SGD methods?

Applying Expected Bound Recursively (Coordinate Optimization)

- Our guaranteed progress bound for randomized coordinate optimization,

$$\mathbb{E}[f(w^{k+1}))] \leq f(w^k) - \frac{1}{2dL}\|\nabla f(w^k)\|^2.$$

- If we subtract f^* and use strong-convexity or PL (as before),

$$\mathbb{E}[f(w^{k+1})] - f^* \leq \left(1 - \frac{\mu}{dL}\right)[f(w^k) - f^*].$$

- By recursing we get linear convergence rate,

$$\begin{aligned}
\mathbb{E}[\mathbb{E}[f(w^{k+1})]] - f^* &\leq \mathbb{E}\left[\left(1 - \frac{\mu}{dL}\right)[f(w^k) - f^*]\right] && \text{(expectation wrt } j_{k-1}\text{)} \\
\mathbb{E}[f(w^{k+1})] - f(w^*) &\leq \left(1 - \frac{\mu}{dL}\right)[\mathbb{E}[f(w^k)] - f^*] && \text{(iterated expectations)} \\
&\leq \left(1 - \frac{\mu}{dL}\right)^2 [f(w^{k-1}) - f^*]
\end{aligned}$$

- You keep alternating between taking an expectation back in time and recursing.

Random Iterate for Non-Convex Rate not depending on Min

- The non-convex SGD bound, but dividing both sides by $\sum_{k=0}^{t} \alpha_k$,

$$\frac{\sum_{k=1}^{t} \alpha_{k-1} \mathbb{E}\|\nabla f(w^{k-1})\|^2}{\sum_{k=0}^{t-1} \alpha_k} \leq \frac{\sum_{k=1}^{t} [\mathbb{E} f(w^{k-1}) - \mathbb{E} f(w^k)] + \sum_{k=1}^{t} \alpha_{k-1}^2 \frac{L\sigma^2}{2}}{\sum_{k=0}^{t-1} \alpha_k}$$

- Now choose $\hat{k} \in \{0, 1, \ldots, t-1\}$ according to $p(\hat{k}) = \alpha_k / \sum_{i=0}^{t-1} \alpha_i$.
- Notice that LHS above is expectation with respect to $\hat{k}$ of $\mathbb{E}\|\nabla f(w^{\hat{k}-1})\|^2$,

$$\mathbb{E}\|\nabla f(w^{\hat{k}-1})\|^2 \leq \frac{f(w^0) - f^*}{\sum_{k=0}^{t-1} \alpha_k} + \frac{L\sigma^2}{2} \frac{\sum_{k=0}^{t-1} \alpha_k^2}{\sum_{k=0}^{t-1} \alpha_k}.$$

- So choosing random iterate achieves same rate without needing to know the min.

Convergence Rate of SGD

- For analyses of SGD under strong convexity, see:
  - Constant α_k: `http://circle.ubc.ca/bitstream/handle/2429/50358/stochasticGradientConstant.pdf`.
  - Decreasing α_k: `http://arxiv.org/pdf/1212.2002v2.pdf`.
- For both cases under PL, see Theorem 4 here:
  - https://arxiv.org/pdf/1608.04636v2.pdf