Numerical Optimization for Machine Learning

How many iterations of gradient descent do we need?

Mark Schmidt

University of British Columbia

Summer 2022

Motivation: Training == Optimization (Usually)

- In machine learning, training is typically written as an optimization problem.

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

- We have a parameter vector w containing d parameters.
- We want to minimize a loss function f, measuring how well we fit the data.
  - We try to minimize f in order to fit the data as well as possible.

- Examples:
  - Linear regression, logistic regression, SVMs, PCA, graphical models, neural nets,...

- Piazza for class-related discussions:
  `piazza.com/ubc.ca/summer2022/cpsc5xx`

Example: Linear Regression

- Example: linear regression with the squared loss,

$$\hat{w} \in \underset{w \in \mathbb{R}^d}{\text{argmin}} \frac{1}{2} \sum_{i=1}^{n} (w^T x^i - y^i)^2,$$

where we have n training examples:
  - Each has a feature vector $x^i \in \mathbb{R}^d$ and label $y^i \in \mathbb{R}$.

- We sometimes write the linear regression objective using matrix notation,

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

where vector $y \in \mathbb{R}^n$ has the labels y^i,
and matrix $X \in \mathbb{R}^{n \times d}$ has the feature vectors $(x^i)^T$ as rows.

Default Tool: Gradient Descent

- Challenges:
  - Often do not have a closed-form solution.
  - Number of parameters d may be large.
  - Evaluating objective f may be expensive.

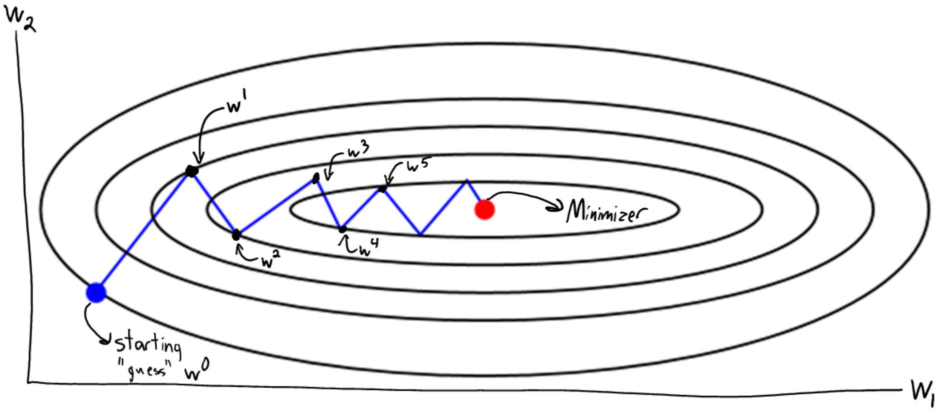
- A common approach is gradient descent.
  - Iterative approach that generates a sequences of guesses w^0, w^1,...
  - First-order method since it requires first derivatives.
  - Cheap iterations: only costs $O(d)$ on top of cost of computing gradient.
  - But, how many iterations do we need?

Outline

Gradient Descent for Finding a Local Minimum

- A typical gradient descent algorithm (first proposed by Cauchy in 1847):
  - Start with some initial guess, w^0.
  - Generate new guess w^1 by moving in the negative gradient direction:
    $$w^1 = w^0 - \alpha_0 \nabla f(w^0),$$
    where α_0 is the step size.
  - Repeat to successively refine the guess:
    $$w^{k+1} = w^k - \alpha_k \nabla f(w^k), \quad \text{for } k = 1, 2, 3, \dots$$
    where we might use a different step-size α_k on each iteration.
  - Stop if $\|\nabla f(w^k)\| \leq \epsilon$.
    - In practice, you also stop if you detect that you are not making progress.
    - Might check $\|w^k - w^{k-1}\|$ or $f(w^{k-1}) - f(w^k)$ for lack of progress.
    - Might also stop if you reach a specified maximum number of iterations.

Gradient Descent in 2D



(Negative gradient direction should be perpendicular to level curves above.)

Gradient Descent for Linear Regression

- The linear regression objective in matrix notation is

$$f(w) = \frac{1}{2}\|Xw - y\|^2.$$

- The gradient vector and Hessian matrix of this objective are

$$\nabla f(w) = X^T(\underbrace{Xw - y}_{r}), \quad \nabla^2 f(w) = \underbrace{X^T X}_{d \times d}.$$

- Plugging the gradient vector into the gradient descent update gives

$$\begin{aligned} w^{k+1} &= w^k - \alpha_k \nabla f(w^k) \\ &= w^k - \alpha_k X^T r. \end{aligned}$$

- Cost of the update is $O(nd)$, with bottleneck being Xw and $X^T r$ operations.
  - If X has only z non-zeroes, cost is reduced to $O(z)$.

Gradient Descent for Regularized Linear Regression

- The L2-regularized linear regression objective in matrix notation is

$$f(w) = \frac{1}{2}\|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2.$$

- The gradient vector and Hessian matrix of this objective are

$$\nabla f(w) = X^T\underbrace{(Xw - y)}_{r} + \lambda w, \quad \nabla^2 f(w) = \underbrace{X^T X}_{d \times d} + \lambda I.$$

- Plugging the gradient vector into the gradient descent update gives

$$\begin{aligned} w^{k+1} &= w^k - \alpha_k \nabla f(w^k) \\ &= w^k - \alpha_k (X^T r + \lambda w^k). \end{aligned}$$

- Cost is the same $O(nd)$ as without regularization.
  - But we will see that you will need fewer iterations.

Cost of L2-Regularizd Least Squares

- Two strategies from 340 for L2-regularized linear regression:
  1. Run t iterations of gradient descent,

$$w^{k+1} = w^k - \alpha_k \underbrace{(X^T(Xw^k - y) + \lambda w^k)}_{\nabla f(w^k)},$$

     which costs $O(ndt)$.
     - Using t as total number of iterations, and k as iteration number.
  2. Closed-form solution by setting gradient equal to zero

$$w = (X^T X + \lambda I)^{-1}(X^T y),$$

     which costs $O(nd^2 + d^3)$.
     - This is fine for $d = 5000$, but may be too slow for $d = 1,000,000$.

- Gradient descent is faster if t is not too big:
  - If we only need $t < \max\{d, d^2/n\}$ iterations.

Cost of Logistic Regression

- Gradient descent can also be applied to other models like logistic regression,

$$f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^i w^T x^i)),$$

  where minimization cannot be formulated as a linear system.
  - Setting $\nabla f(w) = 0$ gives a system of transcendental equations.
- But this objective function is convex and differentiable.
  - So gradient descent converges to a global optimum.
- Alternately, another common approach is Newton's method.
  - Requires computing Hessian $\nabla^2 f(w^k)$, and known as "IRLS" in statistics.

Cost of Logistic Regression

- Gradient descent costs $O(nd)$ per iteration for logistic regression.
- Newton costs $O(nd^2 + d^3)$ per iteration to compute and invert $\nabla^2 f(w^k)$.

- Newton typically requires substantially fewer iterations.

- But for datasets with very large d, gradient descent might be faster.
  - If $t < \max\{d, d^2/n\}$ then we should use the "slow" algorithm with fast iterations.

- So, how many iterations t of gradient descent do we need?

Lipschitz Conintuity of the Gradient

- To understand gradient descent, we will first show a basic property:
  - If the step-size α_k is small enough, then gradient descent decreases f.
- We will assume that the gradient of f is Lipschitz continuous.
  - There exists an L such that for *all* w and v we have

$$\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|.$$

  - "Gradient cannot change arbitrarily fast".
- This holds for many ML models, like least squares and logistic regression.
  - But does not hold for others like matrix factorization or neural networks.
- Assumption is not necessary to show gradient descent decreases f for small α_k.
  - You only need continuity of the gradient, but that is not enough to prove rates.

Lipschitz Continuity of the Gradient

- For C^2 functions, Lipschitz continuity of the gradient is equivalent to

$$\nabla^2 f(w) \preceq LI,$$

  for all w.
- Equivalently: "singular values of the Hessian are bounded above by L".
  - For least squares, minimum L is the maximum eigenvalue of $X^T X$.

- This means we can bound quadratic functions involving the Hessian using

$$\begin{aligned} d^T \nabla^2 f(u) d &\leq d^T (LI) d \\ &= L d^T d \\ &= L\|d\|^2. \end{aligned}$$

Descent Lemma

- For a C^2 function, a variation on the multivariate Taylor expansion is that

$$f(v) = \underbrace{f(w) + \nabla f(w)^T(v-w)}_{\text{tangent hyper-plane}} + \underbrace{\frac{1}{2}(v-w)^T\nabla^2 f(u)(v-w)}_{\text{quadratic function of } v},$$

  for any w and v (with u being some point between w and v).

- Lipschitz continuity implies the green term is at most $L\|v-w\|^2$,

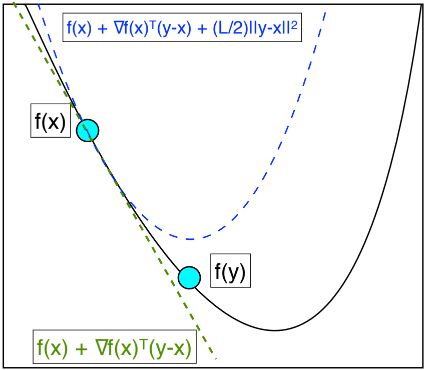
$$f(v) \leq f(w) + \nabla f(w)^T(v-w) + \frac{L}{2}\|v-w\|^2,$$

  which is called the descent lemma.

- The descent lemma also holds for C^1 functions (bonus slide).

Descent Lemma

- The descent lemma gives us a convex quadratic upper bound on f:



- This bound is minimized by a gradient descent step from w with $\alpha_k = 1/L$.

Gradient Descent decreases f for $\alpha_k = 1/L$

- So let us consider doing gradient descent with a step-size of $\alpha_k = 1/L$,

$$w^{k+1} = w^k - \frac{1}{L}\nabla f(w^k).$$

- If we substitle w^{k+1} and w^k into the descent lemma we get

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

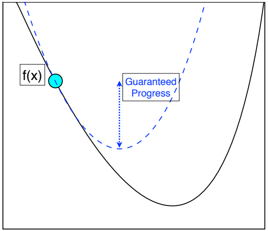
- Now if we use that $(w^{k+1} - w^k) = -\frac{1}{L}\nabla f(w^k)$ in gradient descent,

$$\begin{aligned} f(w^{k+1}) &\leq f(w^k) - \frac{1}{L}\nabla f(w^k)^T\nabla f(w^k) + \frac{L}{2}\|\frac{1}{L}\nabla f(w^k)\|^2 \\ &= f(w^k) - \frac{1}{L}\|\nabla f(w^k)\|^2 + \frac{1}{2L}\|\nabla f(w^k)\|^2 \\ &= f(w^k) - \frac{1}{2L}\|\nabla f(w^k)\|^2. \end{aligned}$$

Implication of Lipschitz Continuity

- We have derived a bound on guaranteed progress when using $\alpha_k = 1/L$.

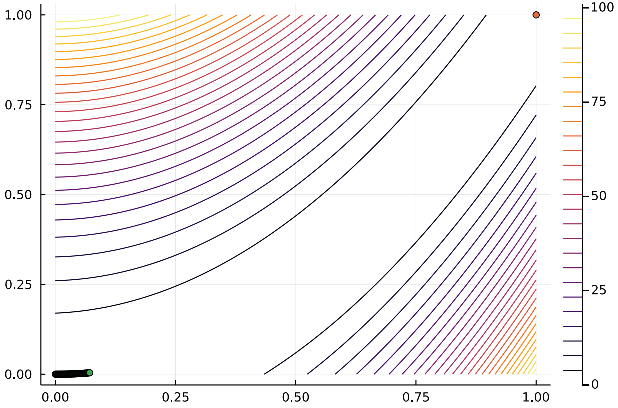
$$f(w^{k+1}) \leq f(w^k) - \frac{1}{2L}\|\nabla f(w^k)\|^2.$$



- If gradient is non-zero, $\alpha_k = 1/L$ is guaranteed to decrease objective.
- Amount we decrease grows with the size of the gradient.

Example: Fixed-Step Gradient Descent on Rosenbrock Function

- Applying gradient descent with $\alpha_k = 1/L$ to Rosenbrock function:



- Red point is optimal solution, green points are 50 iterations of gradient descent.
  - Each iteration decreases function, but the steps are small ($L > 1300$ on $[0, 1]^2$).

Outline

General Step-Size

- Now consider doing using generic step-size of α_k,

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

- We can also derive this step as minimizing a quadratic approximation of f,

$$w^{k+1} \in \underset{w}{\text{argmin}} \left\{ f(w^k) + \nabla f(w^k)^T (w - w^k) + \frac{1}{2\alpha_k} \|w - w^k\|^2 \right\},$$

which is not necessarily an upper bound (since we could have $1/\alpha_k \ll L$).

- Plugging in a generic step-size into the descent lemma and simplifying gives

$$f(w^{k+1}) \leq f(w^k) - \alpha_k \underbrace{\left(1 - \frac{\alpha_k L}{2}\right)}_{> 0 \text{ if } \alpha_k < 2/L} \|\nabla f(w^k)\|^2,$$

which shows that any $\alpha_k < 2/L$ is guaranteed to decrease f.

Do you need to know L?

- In practice, you should never use $\alpha_k = 1/L$.
  - L is usually expensive to compute, and this step-size can be really small.
    - You only need a step-size this small in the worst case (assuming initial $\hat{L} < L$).

- A more-practical option is to approximate L:
  - Start with a small guess for $\hat{L}$ (like $\hat{L} = 1$).
  - On each iteration, before you take your step, check if the progress bound is satisfied:

$$f(\underbrace{w^k - (1/\hat{L})\nabla f(w^k)}_{\text{potential } w^{k+1}}) \leq f(w^k) - \frac{1}{2\hat{L}}\|\nabla f(w^k)\|^2.$$

  - Double $\hat{L}$ if inequality not satisfied, and then test again, until inequality is satisfied.
    - On the next iteration you initialize with the final $\hat{L}$ (so $\hat{L}$ does not decrease).

Do you need to know L?

- "Double $\hat{L}$ until you guarantee enough progress" still guarantees

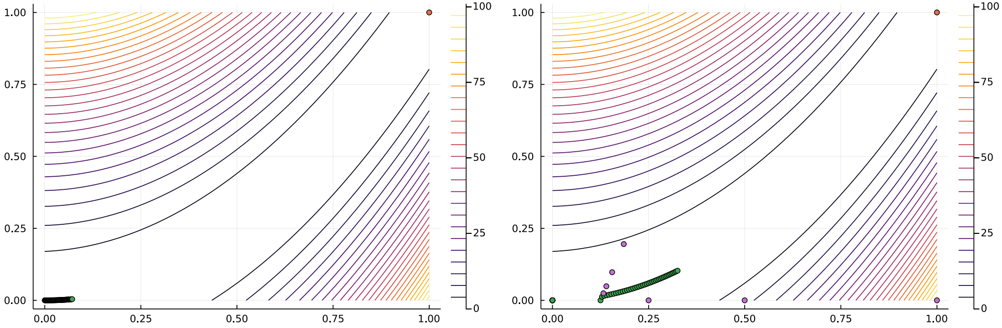
$$f(w^{k+1}) \leq f(w^k) - \frac{1}{4L}\|\nabla f(w^k)\|^2,$$

  in the worst case.
- Because eventually you could double $\hat{L}$ enough so that $L < \hat{L} < 2/L$.
    - And using $\alpha = 1/\hat{L}$ will always decrease f at every iteration.
    - So in the worst case, you guarantee half as much progress as knowing L.
        - Without needing to solve any eigenvalue problems.

- But typically using the estimate $\hat{L}$ works much better than using L.
    - You may only need a step as small as $1/L$ for one direction at one point in space.
    - Typically $\hat{L} \ll L$ so you make way more progress.

Example: Knowing L vs. Aproximating L

- Comoparing $\alpha_k = 1/L$ (left) and $\alpha_k = 1/\hat{L}$ (right).



- Purple points are rejected steps when $\hat{L}$ was too small.
  - After second step $\hat{L} = 256$ (while $L > 1300$ is needed guarantee progress anywhere).

Armijo Backtracking - Textbook Version

- The "double $\hat{L}$" strategy is common for analyzing optimization algorithms.
- But fastest codes typically do not use this strategy.
  - On most iterations, you can make more progress with a bigger step size.
  - You want to exploit the "local" L value, not be slowed down by the global L.

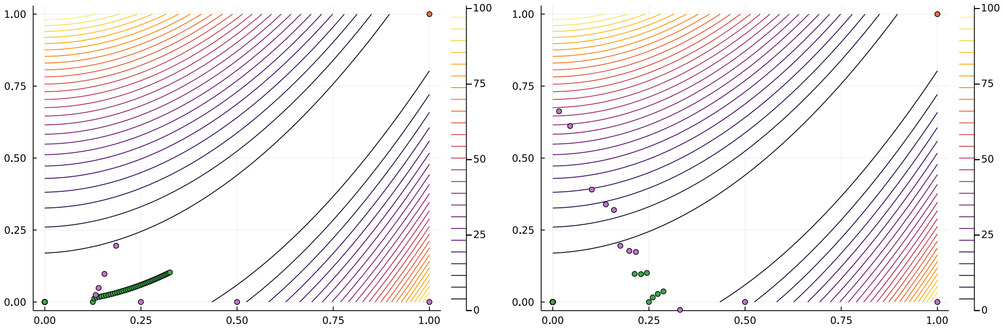
- An approach that usually works better is a backtracking line-search:
  - Start each iteration with a large step-size α.
    - So even if we took small steps in the past, be optimistic that we are not in worst case.
  - Halve α until Armijo condition is satisfied,

$$f(\underbrace{w^k - \alpha \nabla f(w^k)}_{\text{potential } w^{k+1}}) \leq f(w^k) - \alpha\gamma\|\nabla f(w^k)\|^2 \quad \text{for} \quad \gamma \in (0, 1/2],$$

  often we choose γ to be very small like $\gamma = 10^{-4}$.
    - We would rather take a small decrease than try many α values.

Example: Aproximating L vs. Armijo (starting from 1)

- Comoparing $\alpha_k = 1/\hat{L}$ (left) and Armijo starting with $\alpha_k = 1$ and using $\gamma = 10^{-4}$ (right).



- Armijo often allows larger step sizes.
  - Left eventually used $\alpha_k = 1/256$, right used $\alpha_k = 1/128$ (but more backtracking).

Backtracking Line-Search and Armijo Condition

- Good optimization codes use clever tricks to decide on the first α to try.
  - For the first step, many codes initialize with something like $\alpha_0 = 1/\|\nabla f(w^0)\|$.
  - A common choice for subsequent steps is the Barzilai Borwein step-size (next time).
  - Goal of the above is avoid needing to backtrack on most iterations.

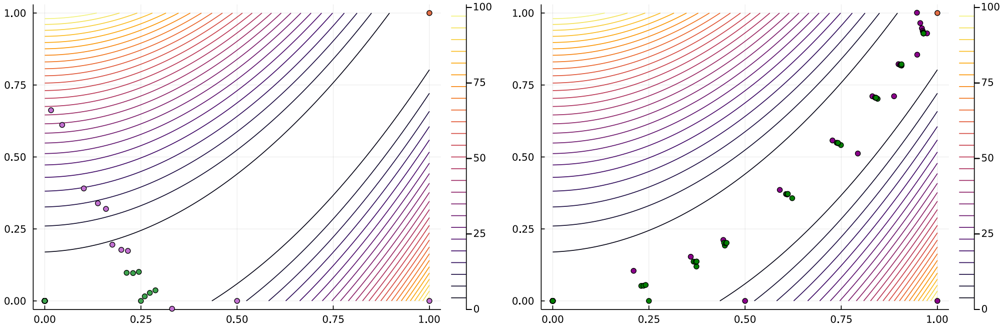
- Good optimization codes use interpolation to decide on subsequent α.
  - Fit polynomial that agrees with observed function and directional derivative values.
  - Set step size to minimize this polynomial.
  - Often gives a step size that is accepted, on iterations where we backtrack.

- Line-search initialization/interpolation help but need safeguards.
  - Sanity checks that they return reasonable values (so do not lose convergence).

- More fancy line-searches are based on the Wolfe conditions.
  - Tests whether directional derivative is decreased along gradient direction.
  - Unlike Armijo, rejects step sizes that are too small.
  - Good reference on these tricks: Nocedal and Wright's Numerical Optimization book.

Example: Armijo (starting from 1) vs. Armijo (starting from BB)

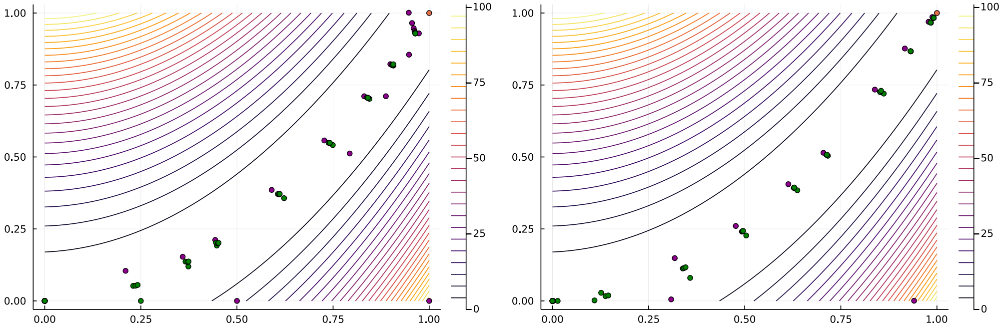
- Armijo starting with 1 (left) and starting from Barzilai-Borwein (right)



- Barzilai-Borwen lead to a lot less backtracking.
  - Accepted step sizes starting from BB ranged from $\approx .0001$ up to ≈ 1.3.

Example: Armijo+BB with step size halving vs. quadratic interpolation

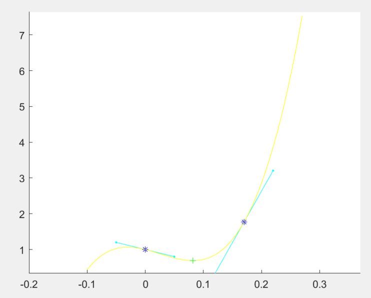
- Armijo + BB initialization, and α_k halving (left) vs. interpolation (right).



- Quadratic interpolation gave more clever guesses when backtracking.
  - *findMin*: gradient descent, Armijo, Barzilai-Browein, quad interp, safeguards.

Example: Backtracking with Polynomial Interpolation

- Backtracking with cubic-polynomial interpolation after rejecting a step size:

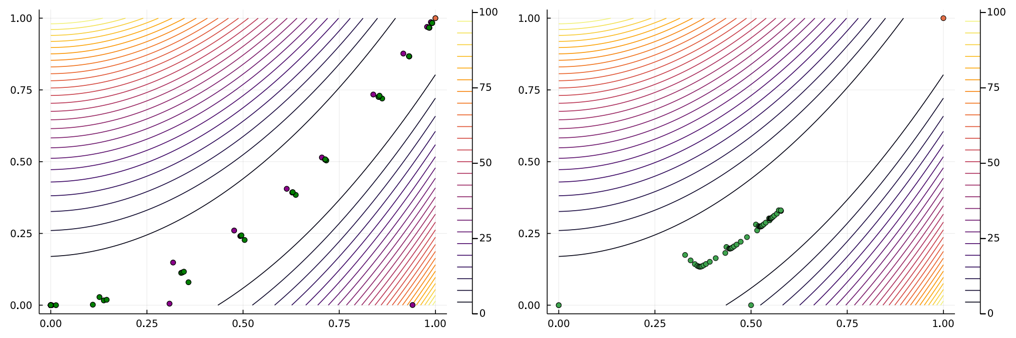


1. Finds cubic polynomial interpolating function agreeing with directional derivatives.
   - $f(w^k)$, $f(w^+)$, $\nabla f(w^k)^T d^k$, $\nabla f(w^+)^T d^k$ for new iteration $w^+ = w^k + \alpha_k d^k$.
   - For gradient descent we have $d^k = -\nabla f(w^k)$.
2. Use the minimum of the cubic polynomial as the step size.

- Quadratic interpolation: two function values and one directional derivative.
  - With n function/derivative values you can fit an $(n-1)$-dimensional polynomial.

Example: Armijo+BB+Quadratic Interp vs. Malitsky-Mischenko

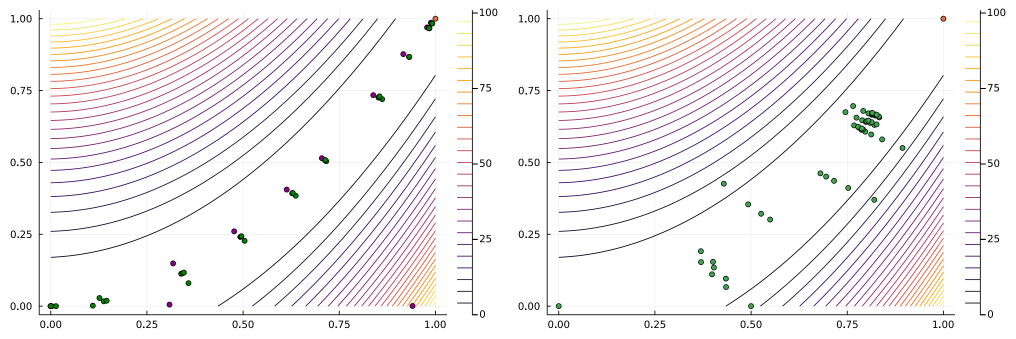
- Comparing Armijo+BB+Quadratic Interp (left) to Malitsky-Mischenko's step size (right):



- I used the initialization suggested in the MM paper ($\alpha_0 = 10^{-10}$).
  - Malitsky-Mischenko does not need backtrack for convex functions.

Example: Armijo+BB+Quadratic Interp vs. Polyak Step Size

- Comparing Armijo+BB+Quadratic Interp (left) to Polyak's step size (right):



- Polyak's step size is $(f(w^k) - f^*)/\|\nabla f(w^k)\|^2$ (assumes f^* is known).
  - Polyak's step size does not need backtrack for convex functions.

Outline

Last Section: Progress Bound for Gradient Descent

- We discussed gradient descent,

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

assuming that the gradient was Lipschitz continuous,

$$\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|,$$

- We showed that setting $\alpha_k = 1/L$ gives a progress bound of

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

- We discussed practical α_k values that give similar bounds (replacing $1/2L$).
  - "Try a big step-size, and decrease it if it does decrease function enough."

Convergence Rate of Gradient Descent

- In 340, we claimed that $\nabla f(w^k)$ converges to zero as k goes to ∞.
  - For convex functions, this means it converges to a global optimum.
  - However, we may not have $\nabla f(w^k) = 0$ for any finite k.

- Instead, we are usually happy with $\|\nabla f(w^k)\| \leq \epsilon$ for some small ϵ.
  - Given an ϵ, how many iterations does it take for this to happen?

- We will first answer this question only assuming that
  1. Gradient ∇f is Lipschitz continuous (as before).
  2. Step-size $\alpha_k = 1/L$ (this is only to make things simpler).
  3. Function f cannot go below a certain value f^* ("bounded below").

- Most ML objectives f are bounded below (like the squared error being at least 0).
  - We're not assuming convexity (but only showing convergence to a stationary point).

Convergence Rate of Gradient Descent

- Key ideas:
  1. We start at some $f(w^0)$, and at each step we decrease f by at least $\frac{1}{2L}\|\nabla f(w^k)\|^2$.
  2. But we cannot decrease $f(w^k)$ below f^*.
  3. So $\|\nabla f(w^k)\|^2$ must be going to zero "fast enough".

- Let's start with our guaranteed progress bound,

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

- Since we want to bound $\|\nabla f(w^k)\|$, let's rearrange as

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

Convergence Rate of Gradient Descent

- So for each iteration k, we have

$$\|\nabla f(w^{k-1})\|^2 \leq 2L[f(w^{k-1}) - f(w^k)].$$

- Consider the smallest squared gradient norm we see before iteration t,

$$\min_{j \in \{0,\ldots,t-1\}} \left\{ \|\nabla f(w^j)\|^2 \right\}.$$

- This minimum is less than the average squared gradient norm,

$$\begin{aligned}
\min_{j \in \{0,\ldots,t-1\}} \left\{ \|\nabla f(w^j)\|^2 \right\} &\leq \frac{1}{t} \sum_{k=1}^{t} \|\nabla f(w^{k-1})\|^2 \\
&\leq \frac{2L}{t} \sum_{k=1}^{t} [f(w^{k-1}) - f(w^k)],
\end{aligned}$$

and each term in average is bounded by the inequality at the top of the slide.

Convergence Rate of Gradient Descent

- The inequality from the previous slide is

$$\min_{j\in\{0,\ldots,t-1\}}\{\|\nabla f(w^j)\|^2\} \leq \frac{2L}{t}\sum_{k=1}^{t}[f(w^{k-1}) - f(w^k)].$$

- On the right, we have a telescoping sum:

$$\sum_{k=1}^{t}[f(w^{k-1}) - f(w^k)] = f(w^0) - \underbrace{f(w^1) + f(w^1)}_{0} - \underbrace{f(w^2) + f(w^2)}_{0} - \cdots - f(w^t)$$
$$= f(w^0) - f(w^t).$$

- And using that $f(w^t) \geq f^*$ we get

$$\min_{k\in\{0,1,\ldots,t-1\}}\left\{\|\nabla f(w^k)\|^2\right\} \leq \frac{2L[f(w^0) - f^*]}{t} = O(1/t),$$

so if we run for t iterations, we'll find $\underbrace{\text{at least one } k}_{\text{the minimum}}$ with $\|\nabla f(w^k)\|^2 = O(1/t)$.

Discussion of $O(1/t)$ Result

- We showed that after t iterations, there is always a k such that

$$\|\nabla f(w^k)\|^2 \leq \frac{2L[f(w^0) - f^*]}{t}.$$

- It isn't necessarily the last iteration t that achieves this.
  - Though iteration t does have the lowest value of $f(w^k)$.
- This is a non-asymptotic result:
  - It is valid for any $t \geq 1$, there is no "limit as $t \to \infty$" as in classic results.
  - But if t goes to ∞, argument can be modified to show that $\nabla f(w^t)$ goes to zero.
    - More precisely, it is common to show that $\liminf_{t\to\infty} \|\nabla f(w^t)\| = 0$.
- This does not imply that gradient descent finds global minimum.
  - We could be minimizing an NP-hard function with bad local optima.

Convergence Rate of Gradient Descent

- Our "error on iteration t" bound:

$$\min_{k \in \{0,1,\dots,t-1\}} \left\{ \|\nabla f(w^k)\|^2 \right\} \leq \frac{2L[f(w^0) - f^*]}{t}.$$

- We want to know when the norm is below ϵ, which is guaranteed if:

$$\frac{2L[f(w^0) - f^*]}{t} \leq \epsilon.$$

- Solving for t gives that this is guaranteed for every t where

$$t \geq \frac{2L[f(w^0) - f^*]}{\epsilon},$$

so gradient descent requires $t = O(1/\epsilon)$ iterations to achieve $\|\nabla f(w^k)\|^2 \leq \epsilon$.

Iteration Complexity

- We have shown that having

$$t \geq \frac{2L[f(w^0) - f^*]}{\epsilon} = O(1/\epsilon),$$

  is sufficient to guarantee we have a k with $\|\nabla f(w^k)\|^2 \leq \epsilon$.

- We say that $t = O(1/\epsilon)$ is the iteration complexity of the algorithm.
  - "Iteration complexity" is "how many you need".
  - Practical step-size strategies like Armijo backtracking also require $O(1/\epsilon)$.
    - Just changes constants inside $O()$ notation.
- For real ML problems iteration complexities are often very loose.
  - In practice gradient descent converges much faster.
  - There is a practical and theoretical component to developing optimization methods.

Outline

Cost of Iterative Algorithms

- Cost of many optimization algorithm is a product of:
  - Cost of each iteration.
  - Iteration Complexity (number of iterations).

- For least squares:
  - Cost of each iteration of gradient descent is $O(nd)$.
  - The iteration complexity we derived is $O(1/\epsilon)$.

  So the total cost according to the result we derived is $O(nd(1/\epsilon))$.

- For a fixed accuracy ϵ:
  - There is dimension d beyond which gradient descent is faster than normal equations.
    - This is assuming that L and $f(w^0) - f^*$ are not growing with the dimensionality.

$O(1/\epsilon)$ vs. $O(\log(1/\epsilon))$

- Think of $\log(1/\epsilon)$ as "number of digits of accuracy" you want.
  - We want iteration complexity to grow slowly with $1/\epsilon$.
- Is $O(1/\epsilon)$ a good iteration complexity?
- Not really, if you need 10 iterations for a "digit" of accuracy then:
  - You might need 100 for 2 digits.
  - You might need 1000 for 3 digits.
  - You might need 10000 for 4 digits.
- We would normally call this exponential time.

Rates of Convergence

- A way to measure rate of convergence is by limit of the ratio of successive errors,

$$\lim_{k \to \infty} \frac{f(w^{k+1}) - f(w^*)}{f(w^k) - f(w^*)} = \rho.$$

- Different ρ values of give us different rates of convergence:
  1. If $\rho = 1$ we call it a sublinear rate.
  2. If $\rho \in (0, 1)$ we call it a linear rate.
  3. If $\rho = 0$ we call it a superlinear rate.

- Having $f(w^t) - f(w^*) = O(1/t)$ gives sublinear convergence rate:
  - "The longer you run the algorithm, the less progress it makes".

Sub/Superlinear Convergence vs. Sub/Superlinear Cost

- As a computer scientist, what would we ideally want?
  - Sublinear rate is bad, we don't want $O(1/t)$ ("exponential" time: $O(1/\epsilon)$ iterations).
  - Linear rate is ok, we're ok with $O(\rho^t)$ ("polynomial" time: $O(\log(1/\epsilon))$ iterations).
  - Superlinear rate is great, amazing to have $O(\rho^{2^t})$ ("constant": $O(\log(\log(1/\epsilon)))$).
- Notice that terminology is backwards compared to computational cost:
  - Superlinear cost is bad, we don't want $O(d^3)$.
  - Linear cost is ok, having $O(d)$ is ok.
  - Sublinear cost is great, having $O(\log(d))$ is great.
- Ideal algorithm: superlinear convergence and sublinear iteration cost.

Polyak-Łojasiewicz (PL) Inequality

- For least squares, we have linear cost but we only showed sublinear rate.
- For many "nice" functions f, gradient descent actually has a linear rate.
- For example, for functions satisfying the Polyak-Łojasiewicz (PL) inequality,

$$\frac{1}{2}\|\nabla f(w)\|^2 \geq \mu(f(w) - f^*),$$

for all w and some $\mu > 0$, and f^* is the infimum of the function.

  - "As the function $f(w)$ increases above f^*, the gradient $\nabla f(w)$ grows quadratically".
  - This property holds for least squares (bonus slide).
  - PL also implies that gradient descent finds global optima.
    - PL implies invexity.
    - For C^1, invexity is equivalent to "all stationary points are global optima".
    - Convex functions a special case of invex functions.

Linear Convergence under the PL Inequality

- Recall our guaranteed progress bound

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

- Under the PL inequality we have $-\|\nabla f(w^k)\|^2 \leq -2\mu(f(w^k) - f^*)$, so

$$f(w^{k+1}) \leq f(w^k) - \frac{\mu}{L}(f(w^k) - f^*).$$

- Let's subtract f^* from both sides,

$$f(w^{k+1}) - f^* \leq f(w^k) - f^* - \frac{\mu}{L}(f(w^k) - f^*),$$

  and factorizing the right side gives

$$f(w^{k+1}) - f^* \leq \left(1 - \frac{\mu}{L}\right)(f(w^k) - f^*).$$

Linear Convergence under the PL Inequality

- Applying this recursively:

$$\begin{aligned} f(w^k) - f^* &\leq \left(1 - \frac{\mu}{L}\right)[f(w^{k-1}) - f(w^*)] \\ &\leq \left(1 - \frac{\mu}{L}\right)\left[\left(1 - \frac{\mu}{L}\right)[f(w^{k-2}) - f^*]\right] \\ &= \left(1 - \frac{\mu}{L}\right)^2 [f(w^{k-2}) - f^*] \\ &\leq \left(1 - \frac{\mu}{L}\right)^3 [f(w^{k-3}) - f^*] \\ &\leq \left(1 - \frac{\mu}{L}\right)^k [f(w^0) - f^*] \end{aligned}$$

- We'll always have $0 < \mu \leq L$ so we have $(1 - \mu/L) < 1$.
  - So PL implies a linear convergence rate: $f(w^k) - f^* = O(\rho^k)$ for $\rho < 1$.

Linear Convergence under the PL Inequality

- We've shown that

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

- By using the inequality that

$$(1 - \gamma) \leq \exp(-\gamma),$$

we have that

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

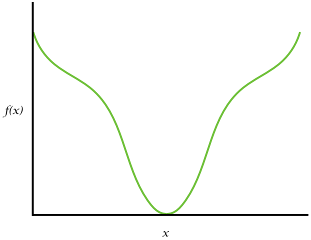
which is why linear convergence is sometimes called "exponential convergence".

- We'll have $f(w^t) - f^* \leq \epsilon$ for any t where

$$t \geq \frac{L}{\mu}\log((f(w^0) - f^*)/\epsilon) = O(\log(1/\epsilon)).$$

Discussion of Linear Convergence under the PL Inequality

- PL is satisfied for many standard convex models like least squares (bonus).
  - So cost of least squares is $O(nd \log(1/\epsilon))$.
- PL is also satisfied for some non-convex functions like $w^2 + 3\sin^2(w)$.



  - PL satisfied for PCA on a certain "Riemann manifold".

Discussion of Linear Convergence under the PL Inequality

- But PL is not satisfied for many models, like neural networks.
- The PL constant μ might be terrible.
  - For least squares μ is the smallest non-zero eigenvalue of the Hessian

.

- It may be hard to show that a function satisfies PL.
- But an important special case of PL functions is strongly-convex functions.
  - Which have many nice properties.

Strong Convexity

- We say that a function f is strongly convex if the function
$$f(w) - \frac{\mu}{2}\|w\|^2,$$
is a convex function for some $\mu > 0$.
  - "If you 'un-regularize' by μ then it's still convex."

- For C^2 functions this is equivalent to assuming that
$$\nabla^2 f(w) \succeq \mu I,$$
that the eigenvalues of the Hessian are at least μ everywhere.

- Some nice properties of strongly-convex functions (see bonus):
  - A unique global minimizing point w^* exists.
  - C^1 strongly-convex functions satisfy the PL inequality.
  - If $g(w) = f(Aw)$ for strongly-convex f and matrix A, then g is PL (least squares).

Strong Convexity Implies PL Inequality

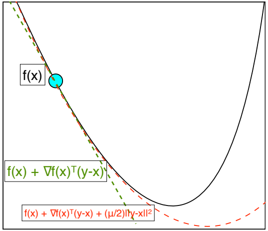
- As before, from Taylor's theorem we have for C^2 functions that

$$f(v) = f(w) + \nabla f(w)^\top (v - w) + \frac{1}{2}(v - w)^\top \nabla^2 f(u)(v - w).$$

- By strong-convexity, $d^\top \nabla^2 f(u) d \geq \mu \|d\|^2$ for any d and u.

$$f(v) \geq f(w) + \nabla f(w)^\top (v - w) + \frac{\mu}{2}\|v - w\|^2$$

- Treating right side as function of v, we get a quadratic lower bound on f.



Strong Convexity Implies PL Inequality

- As before, from Taylor's theorem we have for C^2 functions that

$$f(v) = f(w) + \nabla f(w)^\top (v - w) + \frac{1}{2}(v - w)^\top \nabla^2 f(u)(v - w).$$

- By strong-convexity, $d^\top \nabla^2 f(u) d \geq \mu \|d\|^2$ for any d and u.

$$f(v) \geq f(w) + \nabla f(w)^\top (v - w) + \frac{\mu}{2}\|v - w\|^2.$$

- Treating right side as function of v, we get a quadratic lower bound on f.

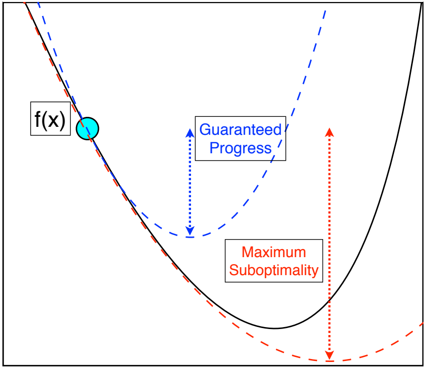
- Minimize both sides in terms of v gives

$$f^* \geq f(w) - \frac{1}{2\mu}\|\nabla f(w)\|^2,$$

which is the PL inequality (bonus slides show for C^1 functions).

Combining Lipschitz Continuity and Strong Convexity

- Lipschitz continuity of gradient gives guaranteed progress.
- Strong convexity of functions gives maximum sub-optimality.



- Progress on each iteration will be at least a fixed fraction of the sub-optimality.

Effect of Regularization on Convergence Rate

- We said that f is strongly convex if the function
$$f(w) - \frac{\mu}{2}\|w\|^2,$$
is a convex function for some $\mu > 0$.
  - For a C^2 univariate function, equivalent to $f''(w) \geq \mu$.
- If we have a convex loss f, adding L2-regularization makes it strongly-convex,
$$f(w) + \frac{\lambda}{2}\|w\|^2,$$
with μ being at least λ.
- So adding L2-regularization can improve rate from sublinear to linear.
  - Go from exponential $O(1/\epsilon)$ to polynomial $O(\log(1/\epsilon))$ iterations.
  - And guarantees a unique minimizer.
    - May not solve unregularized problem.

Effect of Regularization on Convergence Rate

- Our convergence rate under PL was

$$f(w^k) - f^* \leq \underbrace{\left(1 - \frac{\mu}{L}\right)^k}_{\rho^k}[f(w^0) - f^*].$$

- For L2-regularized least squares we have

$$\frac{L}{\mu} = \frac{\max\{\text{eig}(X^\top X)\} + \lambda}{\min\{\text{eig}(X^\top X)\} + \lambda}.$$

- So as λ gets larger ρ gets closer to 0 and we converge faster.

- The number $\frac{L}{\mu}$ is called the condition number of f.
  - For least squares, it's the "matrix condition number" of $\nabla^2 f(w)$.

Summary

- Gradient descent: "repeatedly take small step towards negative gradient".
- Guaranteed progress bound if gradient is Lipschitz, based on norm of gradient.
- Practical step size strategies based on the progress bound.
- Error on iteration t of $O(1/t)$ for functions that are bounded below.
  - Implies that we need $t = O(1/\epsilon)$ iterations to have $\|\nabla f(x^k)\|^2 \leq \epsilon$.
- Sublinear/linear/superlinear convergence measure speed of convergence.
- Polyak-Łojasiewicz inequality leads to linear convergence of gradient descent.
  - Only needs $O(\log(1/\epsilon))$ iterations to get within ϵ of global optimum.
- Strongly-convex differentiable functions satisfy PL-inequality.
  - Adding L2-regularization makes gradient descent go faster.

- Post-lecture slides: Cover various related issues.
  - Checking derivative code, why use gradient direction, descent lemma for C^1.

- Next time: methods that converge faster than gradient descent.

Checking Derivative Code

- Gradient descent codes require you to write objective/gradient code.
  - This tends to be error-prone, although automatic differentiation codes are helping.

- Make sure to check your derivative code:
  - Numerical approximation to partial derivative i:

$$\nabla_i f(x) \approx \frac{f(x + \delta e_i) - f(x)}{\delta},$$

    where δ is small e_i has a 1 at position i and zero elsewhere.
    - Or more accurately using centered differences.
  - For large-scale problems you can check a random direction d:

$$\nabla f(x)^T d \approx \frac{f(x + \delta d) - f(x)}{\delta}$$

  - If the left side coming from your code is very different from the right side, there is likely a bug.

Why the gradient descent iteration?

- For a C^2 function, a variation on the multivariate Taylor expansion is that

$$f(v) = f(w) + \nabla f(w)^T (v - w) + \frac{1}{2}(v - w)^T \nabla^2 f(u)(v - w),$$

for any w and v (with u being some convex combination of w and v).

- If w and v are very close to each other, then we have

$$f(v) = f(w) + \nabla f(w)^T (v - w) + O(\|v - w\|^2),$$

and the last term becomes negligible.

- Ignoring the last term, for a fixed $\|v - w\|$ I can minimize $f(v)$ by choosing $(v - w) \propto -\nabla f(w)$.
  - So if we're moving a small amount the optimal choice is gradient descent.

Descent Lemma for C^1 Functions

- Let ∇f be L-Lipschitz continuous, and define $g(\alpha) = f(x + \alpha z)$ for a scalar α.

$$
\begin{aligned}
f(y) &= f(x) + \int_0^1 \nabla f(x + \alpha(y-x))^T (y-x) d\alpha \quad \text{(fund. thm. calc.)} \\
(\pm \text{ const.}) &= f(x) + \nabla f(x)^T (y-x) + \int_0^1 (\nabla f(x + \alpha(y-x)) - \nabla f(x))^T (y-x) d\alpha \\
(\text{CS ineq.}) &\leq f(x) + \nabla f(x)^T (y-x) + \int_0^1 \|\nabla f(x + \alpha(y-x)) - \nabla f(x)\| \|y-x\| d\alpha \\
(\text{Lipschitz}) &\leq f(x) + \nabla f(x)^T (y-x) + \int_0^1 L\|x + \alpha(y-x) - x\| \|y-x\| d\alpha \\
(\text{homog.}) &= f(x) + \nabla f(x)^T (y-x) + \int_0^1 L\alpha \|y-x\|^2 d\alpha \\
(\int_0^1 \alpha = \tfrac{1}{2}) &= f(x) + \nabla f(x)^T (y-x) + \frac{L}{2}\|y-x\|^2.
\end{aligned}
$$

Equivalent Conditions to Lipschitz Continuity of Gradient

- We said that Lipschitz continuity of the gradient

$$\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|,$$

  is equivalent for C^2 functions to having

$$\nabla^2 f(w) \preceq LI.$$

- There are a lot of other equivalent definitions, see here:
  - `http://xingyuzhou.org/blog/notes/Lipschitz-gradient`.
- An optimization cheat sheet covering [strong-]convexity too is here:
  - `https://fa.bianp.net/blog/2017/optimization-inequalities-cheatsheet/`.

Functions that don't have Lipschitz-continuous Gradient

- A simple example of a function which does not have a Lipschitz-continuous gradient is $f(x) = x^3$: $f''(x) = 6x$ which is not bounded as we vary x.
- Regarding ML applications: any non-smooth function would not have a Lipschitz-continuous gradient, such as the L1-regularizer $g(x) = \lambda\|x\|_1$ or neural networks with ReLU activations. We will discuss non-smooth functions later. For non-smooth functions a common assumption is that that the function (not the gradient) is Lipschitz-continuous.
- Another common type of functions arising in ML that are not Lipschitz-continuous are entropy-like functions. For example, $f(x) = x \log x$ for $x > 0$ is smooth (differentiable) on its domain, but it has $f''(x) = 1/x$ which is not bounded as x approaches 0.
- However, note that $f(x) = x^3$ and $f(x) = x \log x$ (for $x > 0$) have a Lipschitz-continuous gradient over any compact set. So if your iterations stay in a closed and bounded set, then they effectively have a Lipschitz-continuous gradient.
- The way this arises in practice is that if you decrase $f(w^k)$ on each iteration, then the gradient descent iterations w^k stay in the sub-level set $\{w \mid f(w) \leq f(w^0)\}$. You can often show that this set is compact (especially with regularizers that cause the function to grow to ∞ as $\|w\| \to \infty$).

Lipschitz-Continuos Function vs. Lipschitz-Continuous Gradient

- Function f is Lipschitz-cont. if $|f(w) - f(v)| \leq L\|w - v\|$ for some L.
- Gradient ∇f is Lipschitz-cont. if $\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|$ for some L.
  - Some people say that f is "Lipschitz-smooth" here. I avoid to prevent confusing.

- Don't get these confused, and neither implies the other:
  - $f(w) = \|w\|_1$ is Lipschitz-cont. but does not have a Lipschitz-cont. gradient.
  - $f(w) = \frac{1}{2}\|Xw - y\|^2$ is not Lipshitz-cont. but does have a Lipschitz-cont. gradient.
  - $f(w) = \sum_{i=1}^n \log(1 + \exp(-y^i w^\top x^i))$ is Lipschitz-cont. with ∇f Lipschitz-cont.

Why is $\mu \leq L$?

- The descent lemma for functions with L-Lipschitz ∇f is that

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

- Minimizing both sides in terms of v (by taking the gradient and setting to 0 and observing that it's convex) gives

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

- So with PL and Lipschitz we have

$$\frac{1}{2\mu}\|\nabla f(w)\|^2 \geq f(w) - f^* \geq \frac{1}{2L}\|\nabla f(w)\|^2,$$

which implies $\mu \leq L$.

C^1 Strongly-Convex Functions satisfy PL

- If $g(x) = f(x) - \frac{\mu}{2}\|x\|^2$ is convex then from C^1 definition of convexity

$$g(y) \geq g(x) + \nabla g(x)^\top (y - x)$$

or that

$$f(y) - \frac{\mu}{2}\|y\|^2 \geq f(x) - \frac{\mu}{2}\|x\|^2 + (\nabla f(x) - \mu x)^\top (y - x),$$

which gives

$$\begin{aligned} f(y) &\geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2}\|y\|^2 - \mu x^\top y + \frac{\mu}{2}\|x\|^2 \\ &= f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2}\|y - x\|^2, \quad \text{(complete square)} \end{aligned}$$

the inequality we used to show C^2 strongly-convex function f satisfies PL.

Linear Convergence without Strong-Convexity

- The least squares problem is convex but not strongly convex.
  - We could add a regularizer to make it strongly-convex.
  - But if we really want the MLE, are we stuck with sub-linear rates?
- Many conditions give linear rates that are weaker than strong-convexity:
  - 1963: Polyak-Łojasiewicz (PL).
  - 1993: Error bounds.
  - 2000: Quadratic growth.
  - 2013-2015: essential strong-convexity, weak strong convexity, restricted secant inequality, restricted strong convexity, optimal strong convexity, semi-strong convexity.
- Least squares satisfies all of the above.
- Do we need to study any of the newer ones?
  - No! All of the above imply PL except for QG.
  - But with only QG gradient descent may not find optimal solution.

PL Inequality for Least Squares

- Least squares can be written as $f(x) = g(Ax)$ for a σ-strongly-convex g and matrix A, we'll show that the PL inequality is satisfied for this type of function.
- The function is minimized at some $f(y^*)$ with $y^* = Ax$ for some x, let's use $\mathcal{X}^* = \{x | Ax = y^*\}$ as the set of minimizers. We'll use x_p as the "projection" (defined next lecture) of x onto $\mathcal{X}^*$.

$$
\begin{aligned}
f^* = f(x_p) &\geq f(x) + \langle \nabla f(x), x_p - x \rangle + \frac{\sigma}{2}||A(x_p - x)||^2 \\
&\geq f(x) + \langle \nabla f(x), x_p - x \rangle + \frac{\sigma\theta(A)}{2}||x_p - x||^2 \\
&\geq f(x) + \min_y \left[\langle \nabla f(x), y - x \rangle + \frac{\sigma\theta(A)}{2}||y - x||^2 \right] \\
&= f(x) - \frac{1}{2\theta(A)\sigma}||\nabla f(x)||^2.
\end{aligned}
$$

- The first line uses strong-convexity of g, the second line uses the "Hoffman bound" which relies on $\mathcal{X}^*$ being a polyhedral set defined in this particular way to give a constant $\theta(A)$ depending on A that holds for all x (in this case it's the smallest non-zero singular value of A), and the third line uses that x_p is a particular y in the min.

Linear Convergence for "Locally-Nice" Functions

- For linear convergence it's sufficient to have

$$L[f(x^{t+1}) - f(x^t)] \geq \frac{1}{2}\|\nabla f(x^t)\|^2 \geq \mu[f(x^t) - f^*],$$

for all x^t for some L and μ with $L \geq \mu > 0$.

(technically, we could even get rid of the connection to the gradient)

- Notice that this only needs to hold for all x^t, not for all possible x.
  - We could get linear rate for "nasty" function if the iterations stay in a "nice" region.
  - We can get lucky and converge faster than the global L/μ would suggest.

- Arguments like this give linear rates for some non-convex problems like PCA.

Convergence of Function Values vs. Iterates

- Instead of $f(w^k) - f(w^*)$, sometimes we are interested in $\|w^k - w^*\|^2$.
  - Distance from w^k to the nearest global minimum w^* (assuming one exists).
- Under Lipschitz continuity, iterate convergence implies function convergence at same speed.
  - From descent lemma and $\nabla f(w^*) = 0$ we have

$$f(w^k) - f(w^*) \leq \frac{L}{2}\|w^k - w^*\|^2.$$

- Under strong convexity, function convergence implies iterate convergence at same speed.
  - Using the C^1 characterization of strong convexity and $\nabla f(w^*) = 0$ we get

$$f(w^k) - f(w^*) \geq \frac{\mu}{2}\|w^k - w^*\|^2.$$

  - Using the (Lipschitz-gradient)+(strongly-convex) convergence of function values gives

$$\|w^k - w^*\|^2 \leq \frac{\mu}{2} f(w^k) - f(w^*) \leq \frac{\mu}{2}(1 - \mu/L)^k[f(w^0) - f(w^*)].$$

Convergence of Function Values vs. Iterates

- Instead of going through function values, you can directly analyze convergence rate of iterates.
- For (Lipschitz-gradient)+(strongly-convex) with a step-size of $1/L$ you can show:

$$\|w^{k+1} - w^*\| \leq \left(1 - \frac{\mu}{L}\right)\|w^k - w^*\|.$$

- If you use a step-size of $2/(\mu + L)$ this improves to

$$\|w^{k+1} - w^*\| \leq \left(\frac{L - \mu}{L + \mu}\right)\|w^k - w^*\|.$$

- We could convert this back to function values to get a faster rate,

$$f(w^k) - f(w^*) \leq \frac{L}{2}\|w^k - w^*\|^2 \leq \frac{L}{2}\left(\frac{L-\mu}{L+\mu}\right)^{2k}\|w^k - w^*\|^2.$$

Logistic Regression Gradient and Hessian

- With some tedious manipulations, gradient for logistic regression is

$$\nabla f(w) = X^T r.$$

where vector r has $r_i = -y^i h(-y^i w^T x^i)$ and h is the sigmoid function.

- We know the gradient has this form from the multivariate chain rule.
  - Functions for the form $f(Xw)$ always have $\nabla f(w) = X^T r$ (see bonus slide).

- With some more tedious manipulations we get

$$\nabla^2 f(w) = X^T D X.$$

where D is a diagonal matrix with $d_{ii} = h(y_i w^T x^i) h(-y^i w^T x^i)$.
  - The $f(Xw)$ structure leads to a $X^T D X$ Hessian structure.
  - For other problems D may not be diagonal.

Convexity of Logistic Regression

- Logistic regression Hessian is

$$\nabla^2 f(w) = X^T D X.$$

where D is a diagonal matrix with $d_{ii} = h(y_i w^T x^i) h(-y^i w^T x^i)$.

- Since the sigmoid function is non-negative, we can compute $D^{\frac{1}{2}}$, and

$$v^T X^T D X v = v^T X^T D^{\frac{1}{2}} D^{\frac{1}{2}} X v = (D^{\frac{1}{2}} X v)^T (D^{\frac{1}{2}} X v) = \|X D^{\frac{1}{2}} v\|^2 \geq 0,$$

so $X^T D X$ is positive semidefinite and logistic regression is convex.
  - It becomes strictly convex if you add L2-regularization, making solution unique.

Lipschitz Continuity of Logistic Regression Gradient

- Logistic regression Hessian is

$$\nabla^2 f(w) = \sum_{i=1}^n \underbrace{h(y_i w^T x^i) h(-y^i w^T x^i)}_{d_{ii}} x^i (x^i)^T$$

$$\preceq 0.25 \sum_{i=1}^n x^i (x^i)^T$$

$$= 0.25 X^T X.$$

- In the second line we use that $h(\alpha) \in (0,1)$ and $h(-\alpha) = 1 - \alpha$.
    - This means that $d_{ii} \leq 0.25$.
- So for logistic regression, we can take $L = \frac{1}{4}\max\{\text{eig}(X^T X)\}$.