CPSC 540: Machine Learning
Gradient Descent, Newton-like Methods

Mark Schmidt
University of British Columbia

Winter 2017
Admin

- **Auditting/registration forms:**
  - Submit them in class/help-session/tutorial this week.
  - Pick them up in the next class/help-session/tutorial.
  - Add/drop deadline is Tuesday.

- **Tutorials:** start this Friday (4:00 in DMP 110).

- **Assignment 1** due January 16.
  - 1 late day to hand it in January 18.
  - 2 late days to hand it in January 23.
Last Time: MAP Estimation

- We showed that the **loss plus regularizer framework**

\[
    f(w) = \sum_{i=1}^{n} f_i(w) + \lambda g(w),
\]

- can arise from the **MAP estimation principle** applied to IID data,

\[
    w^* \in \arg\max_{w \in \mathbb{R}^d} p(w | y, X) \equiv \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \log p(y^i | x^i, w) - \log p(w).
\]

- **Most common models arise from particular assumptions:**
  - Gaussian likelihood → squared error.
  - Gaussian prior → L2-regularization.
  - Laplace likelihood → absolute error.
  - Sigmoid likelihood → logistic loss.
Least squares corresponds to MLE under the assumption,

$$y^i \sim \mathcal{N}(w^T x^i, \sigma^2),$$

where $\sigma^2$ is irrelevant.

Why does $\sigma^2$ not affect sensitivity to outliers?
- Scales all residuals by the same quantity (unlike switching norms).

If we use a different $\sigma^2_i$ for each example, the $\sigma^2_i$ values would be relevant.
- Leads to weighted least squares

**L2-regularized least squares** corresponds to the assumption

$$y^i \sim \mathcal{N}(w^T x^i, \sigma^2), \quad w_j \sim \mathcal{N}(0, 1/\lambda),$$

with $\sigma^2 = 1$.

Here changing $\sigma^2$ changes solution, but it’s equivalent to changing $\lambda$. 
Last Time: Converting Absolute/Max Problems to Smooth/Constrained

- We turned non-smooth problems involving absolute values and maxes like
  \[
  \arg\min_{w \in \mathbb{R}^d} \|Xw - y\|_1 + \lambda \|w\|_1,
  \]
  into smooth problems with linear constraints,
  \[
  \arg\min_{w \in \mathbb{R}^d, r \in \mathbb{R}^n, v \in \mathbb{R}^d} 1^T r + \lambda 1^T v, \quad \text{with} \quad r \geq Xw - y, \ r \geq y - Xw, \ v \geq w, \ v \geq -w.
  \]
- This is a linear objective and linear constraints: linear program.
- If we had an L2-regularizer or a squared error we would get a quadratic program.
Convex Sets and Functions

- Software like CVX can minimize many convex functions over convex sets.
  - Key property: all local minima are global minima for convex problems.

- We discussed proving sets are convex:
  - Show that for $w$ for $v \in C$, any convex combination $u$ is in $C$.
  - Show that the set is an intersection of convex sets.

- We discussed proving functions are convex:
  - Show that for $w$ for $v \in C$, $f(u)$ is below chord for any convex combination $u$.
  - Show that $\nabla^2 f(w)$ is positive semi-definite for all $w$.
  - Show that $f$ is convex functions and operations that preserve convexity:
    - Non-negative scaling, sum, max, composition with affine map.
Strictly-Convex Functions

- A function is strictly-convex if the convexity definitions hold strictly:
  \[
  f(\theta w + (1 - \theta)v) < \theta f(w) + (1 - \theta)f(v), \quad 0 < \theta < 1 \quad \text{(general)}
  \]
  \[
  f(v) > f(w) + \nabla f(w)^T(v - w) \quad \text{(differentiable)}
  \]
  \[
  \nabla^2 f(w) \succ 0 \quad \text{(twice-differentiable)}
  \]

- Strictly-convex function have at most one global minimum:
  - \(w\) and \(v\) can't be global minima if \(w \neq v\):
    it would imply \(f(u)\) for convex combination \(u\) is below global minimum.

- L2-regularized least squares has unique solution since we showed \(\nabla^2 f(w) \succ 0\).
Outline

1. Gradient Descent Convergence Rate
2. Gradient Descent for Logistic Regression
3. Practical Issues and Newton-Like Methods
Gradient Descent

- Most ML objective functions can’t be written as a linear system/program.
- But many of them yield differentiable and convex objective functions.
  - An example is logistic regression.

- We can minimize these functions using gradient descent:
  - Algorithm for finding a stationary point of a differentiable function.

- Gradient descent is an iterative optimization algorithm:
  - It starts with a “guess” $w^0$.
  - It uses $w^0$ to generate a better guess $w^1$.
  - It uses $w^1$ to generate a better guess $w^2$.
  - ... 
  - The limit of $w^t$ as $t$ goes to $\infty$ has $\nabla f(w^t) = 0$. 
Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Given parameters $w$, the direction of largest instantaneous decrease is $-\nabla f(w)$.
Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Given parameters $w$, the direction of largest instantaneous decrease is $-\nabla f(w)$. 
Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Given parameters $w$, the **direction of largest instantaneous decrease** is $-\nabla f(w)$.

![Gradient descent diagram](image)
Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Given parameters $w$, the direction of largest instantaneous decrease is $-\nabla f(w)$.
Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Given parameters $w$, the direction of largest instantaneous decrease is $-\nabla f(w)$.
Gradient Descent Convergence Rate

Gradient Descent for Logistic Regression

Practical Issues and Newton-Like Methods

Gradient Descent for Finding a Local Minimum

- **Gradient descent** is based on a simple observation:
  - Given parameters $w$, the direction of largest instantaneous decrease is $-\nabla f(w)$.

![Diagram of gradient descent](attachment:image.png)

Now the slope $\nabla f(w^4)$ is positive so we move in the negative direction.
Gradient Descent for Finding a Local Minimum

- **Gradient descent** algorithm:
  - 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 $\alpha^0$ is the **step size**.
  
  - Repeat to **successively refine the guess**:
    \[ w^{t+1} = w^t - \alpha_t \nabla f(w^t), \quad \text{for } t = 1, 2, 3, \ldots \]

  - **Stop** if not making progress $\| \nabla f(w^t) \|$ is small.
  - If $\alpha_t$ is small enough and $\nabla f(w^t) \neq 0$, guaranteed to decrease $f$.
  - Under weak conditions, **procedure converges to a stationary point**.
    - If $f$ is **convex**, converges to a **global minimum**.
Gradient Descent in 2D
Digression: Cost of L2-Regularized Least Squares

- We’ve shown that L2-regularized least squares has the solution

\[ w = (X^T X + \lambda I)^{-1}(X^T y). \]

- With basic matrix multiplication, cost is dominated by:
  - \( O(nd^2) \) to form \( X^T X \).
  - \( O(d^3) \) to solve the linear system.
    - Use “Cholesky” factorization because it’s positive-definite.

- This is fine for \( d = 5000 \), but too slow for \( d = 1,000,000 \).
Cost of L2-Regularized Least Squares

- Would it make any sense to use gradient descent instead?

- The gradient descent iteration would be
  \[ w^{t+1} = w^t - \alpha_t \nabla f(w^t), \quad \text{where} \quad \nabla f(w^t) = X^T(Xw) - X^T y, \]

  and the cost of each iteration is \( O(nd) \), due to the multiplications by \( X \) and \( X^T \).

- So \( t \) iterations of gradient descent cost \( O(ndt) \).

- Gradient descent can be faster if \( t \) is not too big:
  - \( O(ndt) \) is less than \( O(nd^2 + d^3) \) when \( t < \max\{d, d^2/n\} \).
Iteration Complexity

- How many iterations of gradient descent do we need?

- Let $w^*$ be the optimal solution and $\epsilon$ be the accuracy that we want.

- We want to know the smallest number of iteration $t$ that guarantees

$$f(w^t) - f(w^*) \leq \epsilon,$$

which is called the iteration complexity.

- Think of $1/\epsilon$ as “number of digits of accuracy” I want.
  - We want to grow slowly with $1/\epsilon$. 
Strong-Smoothness and Strong-Convexity Assumptions

- We’ll assume $f$ is twice-differentiable and satisfies two assumptions on $\nabla^2 f(w)$:
  - Strong smoothness means that eigenvalues of $\nabla^2 f(w)$ are at most a $L < \infty$
  - Strong convexity means that the eigenvalues of $\nabla^2 f(w)$ are at least $\mu > 0$.

- We denote these assumptions by
  \[
  \mu I \preceq \nabla^2 f(w) \preceq LI, \quad \forall w.
  \]

- Equivalently, for all $w$ and $v$ we have
  \[
  \mu \|v\|^2 \leq v^T \nabla^2 f(w)v \leq L \|v\|^2.
  \]

- Note that strong-convexity $\Rightarrow$ strict-convexity $\Rightarrow$ convexity:
  \[
  \nabla^2 f(w) \succeq \mu I \succ 0 \succeq 0.
  \]

- Strongly-convex functions on closed convex sets have exactly 1 minimizer.
- For L2-regularized least squares we have (see bonus slide).
  \[
  L = \max \{\text{eig}(X^T X)\} + \lambda, \quad \mu = \min \{\text{eig}(X^T X)\} + \lambda,
  \]
We'll use different notation for optimization algorithms:

- For optimization algorithms our variables will be $x$ instead of $w$.

So the gradient descent iteration will be

$$x^{t+1} = x^t - \alpha_t \nabla f(x^t).$$
Convergence Rate of Gradient Descent

- For our first result we’re assuming:
  - Function $f$ is $L$-strongly smooth and $\mu$-strongly convex,
    \[ \mu I \preceq \nabla^2 f(x) \preceq LI. \]
  - We use a step-size of $\alpha_t = 1/L$ (makes proof easier).

- We’ll show that gradient descent has a linear convergence rate,
  \[ f(x^t) - f(x^*) = O(\rho^t) \quad \text{for} \quad \rho < 1. \]
  which is sometimes called “geometric” or “exponential” convergence rate.

- Implies that iteration complexity is $t = O(\log(1/\epsilon))$ iterations (see bonus slide).
  - This is good! We’re growing with logarithm of “digits of accuracy”.

- Gradient Descent Convergence Rate
- Gradient Descent for Logistic Regression
- Practical Issues and Newton-Like Methods
Implication of Strong-Smoothness

- From Taylor’s theorem, for any $x$ and $y$ there is a $z$ such that
  \[
  f(y) = f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} (y - x)^T \nabla^2 f(z) (y - x)
  \]
- By strong-smoothness, $v^T \nabla^2 f(z) v \leq L \|v\|^2$ for any $v$ and $z$.
  \[
  f(y) \leq f(x) + \nabla f(x)^T (y - x) + \frac{L}{2} \|y - x\|^2
  \]
- Treating right side as a function of $y$, we get a quadratic upper bound on $f$. 
Implication of Strong-Smoothness

- The quadratic upper-bound from strong-smoothness at $x^t$ is:

$$f(y) \leq f(x^t) + \nabla f(x^t)^T (y - x^t) + \frac{L}{2} \|y - x^t\|^2$$

- If we set $x^{t+1}$ to minimize the right side in terms of $y$, we get

$$x^{t+1} = x^t - \frac{1}{L} \nabla f(x^t),$$

so gradient descent with $\alpha_t = 1/L$ minimizes this quadratic upper bound.

- Plugging in $x^{t+1}$ gives:

$$f(x^{t+1}) \leq f(x^t) + \nabla f(x^t)^T (x^{t+1} - x^t) + \frac{L}{2} \|x^{t+1} - x^t\|^2$$

$$= f(x^t) - \frac{1}{L} \nabla f(x^t)^T \nabla f(x^t) + \frac{1}{2L} \|\nabla f(x^t)\|^2$$

$$(x^{t+1} - x^t) = -\frac{1}{L} \nabla f(x^t)$$

$$= f(x^t) - \frac{1}{2L} \|\nabla f(x^t)\|^2.$$
Implication of Strong-Smoothness

- We’ve derived a bound on guaranteed progress at iteration $t$:

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

- If gradient is non-zero, guaranteed to decrease objective.
- Amount we decrease grows with the size of the gradient.
- This bound holds for any strongly-smooth function (including non-convex).
Implication of Strong-Convexity

- From Taylor’s theorem, for any $x$ and $y$ there is a $z$ such that

$$f(y) = f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} (y - x)^T \nabla^2 f(z) (y - x)$$

- By strong-convexity, $v^T \nabla^2 f(z) v \geq \mu \|v\|^2$ for any $v$ and $z$.

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

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

![Graph showing quadratic lower bound](image_url)
Implication of Strong-Convexity

- From Taylor’s theorem, for any $x$ and $y$ there is a $z$ such that

$$f(y) = f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} (y - x)^T \nabla^2 f(z) (y - x)$$

- By strong-convexity, $v^T \nabla^2 f(z) v \geq \mu \|v\|^2$ for any $v$ and $z$.

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

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

- Minimize both sides in terms of $y$ gives

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

- This upper bounds how far where we are from the solution.
Combining Strong-Smoothness and Strong-Convexity

Given $x^t$, we have bounds on $f(x^{t+1})$ and $f(x^*)$:

$$f(x^{t+1}) \leq f(x^t) - \frac{1}{2L} \|\nabla f(x^t)\|^2, \quad f(x^*) \geq f(x^t) - \frac{1}{2\mu} \|\nabla f(x^t)\|^2.$$
Combining Strong-Smoothness and Strong-Convexity

- Our bound on guaranteed progress:
  \[
  f(x^{t+1}) \leq f(x^t) - \frac{1}{2L} \|\nabla f(x^t)\|^2.
  \]

- Re-arranging our bound on “distance to go”:
  \[
  -\frac{1}{2} \|\nabla f(x^t)\|^2 \leq -\mu [f(x^t) - f(x^*)].
  \]

- Use “distance to go” bound in guaranteed progress bound:
  \[
  f(x^{t+1}) \leq f(x^t) - \frac{1}{L} \left( \mu [f(x^t) - f(x^*)] \right).
  \]

- Subtract \( f(x^*) \) from both sides and factor:
  \[
  f(x^{t+1}) - f(x^*) \leq f(x^t) - f(x^*) - \frac{\mu}{L} [f(x^t) - f(x^*)]
  = \left( 1 - \frac{\mu}{L} \right) [f(x^t) - f(x^*)].
  \]
Combining Strong-Smoothness and Strong-Convexity

- We’ve shown that

\[ f(x^t) - f(x^*) \leq \left(1 - \frac{\mu}{L}\right) \left[f(x^{t-1}) - f(x^*)\right]. \]

- Applying this recursively:

\[
\begin{align*}
  f(x^t) - f(x^*) &\leq \left(1 - \frac{\mu}{L}\right) \left[\left(1 - \frac{\mu}{L}\right) \left[f(x^{t-2}) - f(x^*)\right]\right] \\
  &= \left(1 - \frac{\mu}{L}\right)^2 \left[f(x^{t-2}) - f(x^*)\right] \\
  &\leq \left(1 - \frac{\mu}{L}\right)^3 \left[f(x^{t-3}) - f(x^*)\right] \\
  &\leq \left(1 - \frac{\mu}{L}\right)^t \left[f(x^0) - f(x^*)\right]
\end{align*}
\]

- Since \( \mu \leq L \), we have \( (1 - \mu/L) < 1 \), and we’ve shown linear convergence rate:
  - We have \( f(x^t) - f(x^*) = O(\rho^t) \) with \( \rho = (1 - \mu/L) \).
Discussion of Linear Convergence Rate

We’ve shown that gradient descent under certain settings has

\[ f(x^t) - f(x^*) \leq \left(1 - \frac{\mu}{L}\right)^t [f(x^0) - f(x^*)]. \]

This is a non-asymptotic linear convergence rate:

- It holds on iteration 1, there is no “limit as \( t \to \infty \)” as in classic results.

The number \( L/\mu \) is called the condition number of \( f \).

- For least squares it’s the “matrix condition number” of the Hessian,
  \[ L/\mu = \text{cond}(\nabla^2 f(w)) = \text{cond}(X^T X). \]

This convergence rate is dimension-independent:

- It does not directly depend on dimension \( d \).
- Though \( L \) might grow and \( \mu \) might shrink as dimension increases.

Consider a fixed condition number and accuracy \( \epsilon \):

- There is a dimension \( d \) beyond which gradient descent is faster than linear algebra.
Outline

1. Gradient Descent Convergence Rate
2. Gradient Descent for Logistic Regression
3. Practical Issues and Newton-Like Methods
Gradient Descent for Logistic Regression

- Is gradient descent useful beyond least squares?
  - Yes: these types of methods tend to work well for a variety of models.

- For example, logistic regression is among most-used models,

\[ f(w) = \sum_{i=1}^{n} \log(1 + \exp(-y^T w x^i)) + \frac{\lambda}{2} \|w\|^2. \]

- We can't even formulate as a linear system or linear program.
  - Setting \( \nabla f(w) = 0 \) gives a system of transcendental equations.

- But this objective function is convex and differentiable.
- Let's compute the cost of minimizing \( f \) with gradient descent.
Gradient Descent for Logistic Regression

To apply gradient descent, we’ll need the gradient.

Can we write logistic loss,

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

in matrix notation?

A “Matlab-y” way:

$$f(w) = 1^T \log(1 + \exp(-Y X w)),$$

where we’re using “element-wise” versions of log and exp function.
Gradient Descent for Logistic Regression

- To write in matrix notation without defining new operators we can use
  \[ f(w) = 1^T v + \frac{\lambda}{2} \|w\|^2 \]
  where \( v_i = \log(1 + \exp(-y_i w^T x^i)) \).

- With some tedious manipulations we get
  \[ \nabla f(w) = X^T r + \lambda w \]
  where \( r_i = -y_i \sigma(-y_i w^T x^i) \).

- We know 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).
Gradient Descent for Logistic Regression

- The gradient has the form

\[ \nabla f(w) = X^T r + \lambda w \]

where \( r_i = -y_i \sigma(-y_i w^T x^i) \).

- The cost of computing the gradient is dominated by:
  1. Computing \( Xw \) to get the \( n \) values \( w^T x^i \).
  2. Computing \( X^T r \) to get the gradient.

- These are matrix-vector multiplications, so the cost is \( O(nd) \).
  - So iteration cost is the same as least squares.
Gradient Descent for Logistic Regression

With some more tedious manipulations we get

$$\nabla^2 f(w) = X^T DX + \lambda I$$

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

- The $f(Ax)$ structure leads to a $X^T DX$ Hessian structure.

- This implies the function is strongly-smooth and strongly-convex with

$$L = \frac{1}{4} \max\{\text{eig}(X^T X)\} + \lambda, \quad \mu = \lambda.$$ 

(1/4 is the maximum value of $d_{ii}$ and the minimum converges to 0.)
Gradient Descent and Logistic Regression

- Condition number $L/\mu$ for L2-regularized least squares was

$$\frac{\max\{\text{eig}(X^T X)\} + \lambda}{\min\{\text{eig}(X^T X)\} + \lambda},$$

while for logistic regression it is

$$\frac{1}{4} \frac{\max\{\text{eig}(X^T X)\} + \lambda}{\lambda}.$$

- So number of iterations for logistic regression is similar to least squares.
- Also, in both cases number of iterations gets smaller as $\lambda$ increases.

- For fixed condition number, total cost is $O(nd\log(1/\epsilon))$.
- Common approach in many software packages is called IRLS:
  - A Newton-like method that takes $O(nd^2 + d^3)$ per iteration.
Outline

1 Gradient Descent Convergence Rate
2 Gradient Descent for Logistic Regression
3 Practical Issues and Newton-Like Methods
Gradient Method: Practical Issues

- In practice, you should never use $\alpha = 1/L$.
  - Often you don’t know $L$, or it's expensive to compute.
  - The “local” $L$ may be much smaller than the “global” $L$.
  - You might also get a “lucky” direction that makes much more progress.
    - In practice, you can often take much bigger steps.

- One practical option is an adaptive step-size:
  - Start with a small guess for $L$ (like $L = 1$).
  - Double $L$ if the progress inequality in the proof is not satisfied:

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

- This often gives you a much smaller $L$: gives bigger steps and faster progress.
- But with this strategy, step-size never increases.
Gradient Method: Practical Issues

- In practice, you should never use $\alpha = 1/L$.
  - Often you don’t know $L$, or it’s expensive to compute.
  - Even if you did, the “local” $L$ may be much smaller than the “global” $L$.
  - You might also get a “lucky” direction that makes much more progress.
    - In practice, you can often take much bigger steps.

- Another practical option is a backtracking line-search:
  - On each iteration, start with a large step-size $\alpha$.
  - Decrease $\alpha$ if the Armijo condition is not satisfied,
    \[
    f(x^{t+1}) \leq f(x^t) - \alpha \gamma \|\nabla f(x^t)\|^2 \quad \text{for} \quad \gamma \in (0, 1/2].
    \]
    (often $\gamma = 10^{-4}$)
  - Tends to work well if you use interpolation to select initial/decreasing $\alpha$ values.
    - Good codes often only need around 1 value of $\alpha$ per iteration.
    - Even more fancy line-search: Wolfe conditions (make sure $\alpha$ is not too small).
Gradient Method: Practical Issues

- Gradient descent codes require you to write objective/gradient code:

  ```matlab
  function [nll,g,H] = objective(w,X,y,lambda)
  yXw = y.*(X*w);
  % Function value
  nll = sum(log(1+exp(-yXw))) + (lambda/2)*(w'*w);
  % Gradient
  sigmoid = 1./(1+exp(-yXw));
  g = -X'*(y.*(1-sigmoid)) + lambda*w;
  ```

- Make sure to check your derivative code:
  - Numerical approximation to partial derivative:
    \[
    \nabla_i f(x) \approx \frac{f(x + \delta e_i) - f(x)}{\delta}
    \]
  - For large-scale problems you can check a random direction \(d\):
    \[
    \nabla f(x)^T d \approx \frac{f(x + \delta d) - f(x)}{\delta}
    \]
Heavy-Ball Method Method

Gradient Method

Heavy-ball Method
Heavy-Ball Method Method

Gradient Method

Heavy-ball Method

$w^0$

$w$

$w^1$
Heavy-Ball Method Method
Heavy-Ball Method Method
Heavy-Ball Method Method
Heavy-Ball Method Method

Gradient Method

Heavy-ball Method

$w^0$

$w$

$w^2$

$w^3$

$w^0$

$w^1$

$w^2$

$w^3$
Heavy-Ball Method Method

Gradient Method

Heavy-ball Method
Heavy-Ball Method Method

Gradient Method

approaches from left

Heavy-ball Method

Bounce around
Heavy-Ball Method and Variations

- The **heavy-ball** method (called **momentum** in neural network papers) is

\[ x^{t+1} = x^t - \alpha_t \nabla f(x^t) + \beta_t (x^t - x^{t-1}). \]

- Faster rate for strictly-convex quadratic functions with appropriate \( \alpha_t \) and \( \beta_t \).
  - Depends on \( \sqrt{L/\mu} \) instead of \( L/\mu \).
  - With the optimal \( \alpha_t \) and \( \beta_t \), we obtain **conjugate gradient**.
    - “Optimal” rate for strongly-convex quadratics in “high-dimensional setting”.

- Variation is **Nesterov’s accelerated gradient method** for strongly-smooth \( f \),

\[
\begin{align*}
  x^{t+1} &= y^t - \alpha_t \nabla f(y^t), \\
  y^{t+1} &= x^t + \beta_t (x^{t+1} - x^t),
\end{align*}
\]

- Rate depends on \( \sqrt{L/\mu} \) for **strongly-convex** \( f \) for appropriate \( \alpha_t \) and \( \beta_t \).

[Note: The image contains a watermark that reads “Gradient Descent Convergence Rate Gradient Descent for Logistic Regression Practical Issues and Newton-Like Methods.”]
Newton’s Method

- Newton’s method is a second-order strategy. (also called IRLS for functions of the form $f(Ax)$)

- Modern form uses the update
  \[ x^{t+1} = x^t - \alpha_t d^t, \]
  where $d^t$ is a solution to the system
  \[ \nabla^2 f(x^t)d^t = \nabla f(x^t). \]  (Assumes $\nabla^2 f(x^t) \succ 0$)

- Equivalent to minimizing the quadratic approximation:
  \[
  f(y) \approx f(x^t) + \nabla f(x^t)^T (y - x^t) + \frac{1}{2\alpha_t} (y - x^t) \nabla^2 f(x^t) (y - x^t).
  \]

- We can generalize the Armijo condition to
  \[
  f(x^{t+1}) \leq f(x^t) + \gamma \alpha \nabla f(x^t)^T d^t.
  \]
  Has a natural step length of $\alpha = 1$. (always accepted when close to a minimizer)
Newton’s Method
Newton’s Method
Newton’s Method

\[
x - \alpha f'(x)
\]
Newton’s Method

The diagram illustrates the concept of Newton's Method for finding the minimum of a function $f(x)$. The method involves the iterative calculation of:

$$x_{n+1} = x_n - \alpha f'(x_n)$$

where $x_n$ is the current point, $f'(x)$ is the derivative of the function at $x$, and $\alpha$ is a scalar parameter that determines the step size. The function $f(x)$ is shown as a solid line, and $Q(x)$ represents the quadratic approximation of $f(x)$ around the current point $x$.
Newton’s Method

\[ f(x) \]

\[ x^k - \alpha H^{-1}f'(x) \]

\[ x \]

\[ Q(x) \]

\[ x - \alpha f'(x) \]
Convergence Rate of Newton’s Method

- If $\mu I \preceq \nabla^2 f(x) \preceq LI$ and $\nabla^2 f(x)$ is Lipschitz-continuous, then close to $x^*$ Newton’s method has local superlinear convergence:
  \[
  f(x^{t+1}) - f(x^*) \leq \rho_t [f(x^t) - f(x^*)],
  \]
  with $\lim_{t \to \infty} \rho_t = 0$.
- Converges very fast, use it if you can!
- But Newton’s method is expensive if dimension $d$ is large:
  - Requires solving $\nabla^2 f(x^t) d^t = \nabla f(x^t)$.
- “Cubic regularization” of Newton’s method gives global convergence rates.
Practical Approximations to Newton’s Method

- **Practical Newton-like** methods (that can be applied to large-scale problems):
  1. **Diagonal** approximation:
     - Approximate Hessian by a **diagonal matrix** \( D \) (cheap to store/invert).
     - A common choice is \( d_{ii} = \nabla^2 f(x^t) \).
     - This sometimes helps, often doesn’t.
  2. **Limited-memory quasi-Newton** approximation:
     - Approximates Hessian by a **diagonal plus low-rank** approximation \( B^t \),
       \[
       B^t = D + UV^T,
       \]
     - which supports fast multiplication/inversion.
     - Based on “quasi-Newton” equations which use differences in gradient values.
       \[
       (\nabla f(x^t) - \nabla f(x^{t-1})) = B^t(x^t - x^{t-1}).
       \]
     - A common choice is **L-BFGS**.
Practical Approximations to Newton’s Method

- **Practical Newton-like methods** (that can be applied to large-scale problems):
  1. **Barzilai-Borwein** approximation:
     - Approximates Hessian by the identity matrix (as in gradient descent).
     - But chooses step-size based on least squares solution to quasi-Newton equations.
     \[ \alpha_t = -\alpha_t \frac{v^T \nabla f(w)}{\|v\|^2}, \quad \text{where} \quad v = \nabla f(x^t) - \nabla f(x^{t-1}). \]
     - Works better than it deserves to (**findMind.m** from CPSC 340).
     - We don’t understand why it works so well.
  2. **Hessian-free Newton**:
     - Uses conjugate gradient to approximately solve Newton system.
     - Requires Hessian-vector products, but these cost same as gradient.
     - If you’re lazy, you can numerically approximate them using
     \[ \nabla^2 f(x^t)d \approx \frac{\nabla f(x^t + \delta d) - \nabla f(x^t)}{\delta}. \]
     - If \( f \) is analytic, can compute exactly by evaluating gradient with complex numbers.
       (look up “complex-step derivative”)
     - A related approach to the above is **non-linear conjugate gradient**.
Numerical Comparison with minFunc

Result after 25 evaluations of limited-memory solvers on 2D rosenbrock:

x1 = 0.0000, x2 = 0.0000 (starting point)
x1 = 1.0000, x2 = 1.0000 (optimal solution)

x1 = 0.3654, x2 = 0.1230 (minFunc with gradient descent)
x1 = 0.8756, x2 = 0.7661 (minFunc with Barzilai-Borwein)
x1 = 0.5840, x2 = 0.3169 (minFunc with Hessian-free Newton)
x1 = 0.7478, x2 = 0.5559 (minFunc with preconditioned Hessian-free Newton)
x1 = 1.0010, x2 = 1.0020 (minFunc with non-linear conjugate gradient)
x1 = 1.0000, x2 = 1.0000 (minFunc with limited-memory BFGS - default)
Summary

- **Gradient descent** is finding stationary point of differentiable $f$.
- **Iteration complexity** measures number of iterations to reach accuracy $\epsilon$.
- **Linear convergence rate** is achieved by gradient descent.
- **Faster first-order methods** like Nesterov and Newton-like methods.

Next time: is using L1-regularization as easy as using L2-regularization?
**Bonus Slide: Constants for Least Squares**

- Consider **least squares**: \( f(x) = \frac{1}{2} \|Ax - b\|^2 \)

  What are 'L' and 'm' such that \( m \leq \nabla^2 f(x) \leq L I \)?

  Note that \( \nabla^2 f(x) = A^T A \) and since it's symmetric we can use **spectral decomposition**:

  \[
  A^T A = \sum_{j=1}^{d} \lambda_j q_j q_j^T \quad \text{where} \quad q_j^T q_j = 1 \quad \text{and} \quad q_i^T q_j = 0 \quad \text{for} \quad i \neq j.
  \]

  We can write any \( y \) as linear combination of orthogonal basis, \( y = \alpha_1 q_1 + \alpha_2 q_2 + \cdots + \alpha_d q_d \).

  So we have \( y^T \nabla^2 f(x) y = y^T A^T A y = y^T (A^T y) q_j = \sum_{j=1}^{d} \lambda_j \alpha_j^2 \).

  Note that we can assume \( \|y\|_1 = 1 \) or \( y^T y = \sum_{j=1}^{d} \alpha_j^2 = 1 \).

  So \( y^T \nabla^2 f(x) y \) is maximized when \( \alpha_j^2 = 1 \) and minimized when \( \alpha_j^2 = 0 \),

  giving \( L = \lambda_1 = \max(\text{eig}(A^T A)) \) and \( m = \lambda_d = \min(\text{eig}(A^T A)) \).
If we have
\[ f(w^t) - f(w^*) = \epsilon = O(\rho^t), \]
this means \( \epsilon \leq \kappa \rho^t \) for some \( \kappa \) for large \( t \) or
\[ \log(\epsilon) \leq \log(\kappa \rho^t) = \log(\kappa) + t \log(\rho), \]
or
\[ t \geq \log(\epsilon)/\log(\rho) - \text{constant}, \]
or that it holds for any
\[ t \geq O(\log(1/\epsilon)) \quad \text{since} \quad \rho < 1. \]

Often \( \rho \) has the form \( (1 - 1/\kappa) \), so if we use \( (1 - 1/\kappa) \leq \exp(-\kappa) \) we get
\[ t \geq O(\kappa \log(1/\epsilon)). \]
Bonus Slide: Multivariate Chain Rule in Matrix Notation

If \( g : \mathbb{R}^d \mapsto \mathbb{R}^n \) and \( f : \mathbb{R}^n \mapsto \mathbb{R} \), then \( h(x) = f(g(x)) \) has gradient

\[
\nabla h(x) = \nabla g(x)^T \nabla f(g(x)),
\]

where \( \nabla g(x) \) is the Jacobian (since \( g \) is multi-output).

If \( g \) is an affine map \( x \mapsto Ax + b \) so that \( h(x) = f(Ax + b) \) then we obtain

\[
\nabla h(x) = A^T \nabla f(Ax + b).
\]

Further, for the Hessian we have

\[
\nabla^2 h(x) = A^T \nabla^2 f(Ax + b) A.
\]
Bonus Slide: Convergence of Gradient Descent

We can show convergence of gradient descent without strong convexity.