Numerical Computation

Sargur N. Srihari
srihari@cedar.buffalo.edu

This is part of lecture slides on Deep Learning:
http://www.cedar.buffalo.edu/~srihari/CSE676

Topics

- Overflow and Underflow
- Poor Conditioning
- Gradient-based Optimization
- Stationary points, Local minima
- Second Derivative
- Convex Optimization
- Lagrangian

Acknowledgements: Goodfellow, Bengio, Courville, Deep Learning, MIT Press, 2016

Overview

- ML algorithms usually require a high amount of numerical computation
  - To update estimate of solutions iteratively
    - not analytically derive formula providing expression
- Common operations:
  - Optimization
    - Determine maximum or minimum of a function
  - Solving system of linear equations
- Just evaluating a mathematical function of real numbers with finite memory can be difficult

Overflow and Underflow

- Problem caused by representing real numbers with finite bit patterns
  - For almost all real numbers we encounter approximations
- Although a rounding error it compounds across many operations and algorithm will fail
  - Numerical errors
    - Underflow: when nos close to zero are rounded to zero
      - $log\ 0$ is -∞ (which becomes not-number for further operations)
    - Overflow: when nos with large magnitude are approximated as -∞ or +∞ (Again become not-no.)

Function needing stabilization for Over/Underflow

- Softmax probabilities in multinoulli

$$\text{softmax}(\boldsymbol{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$$

  - Consider when all x_i are equal to some c. Then all probabilities must equal $1/n$. This may not happen
    - When c is a large negative; denominator =0, result undefined underflow
    - When c is large positive, $\exp(c)$ will overflow
  - Circumvented using $\text{softmax}(\boldsymbol{z})$ where $\boldsymbol{z}=\boldsymbol{x}\text{-}\max_i x_i$
- Another problem: underflow in numerator can cause $\log \text{softmax}(\boldsymbol{x})$ to be $-\infty$
  - Same trick can be used as for softmax

Dealing with numerical consderations

- Developers of low-level libraries should take this into consideration
- ML libraries should be able to provide such stabilization
  - Theano for Deep Learning detects and provides this

Poor Conditioning

- Conditioning refers to how rapidly a function changes with a small change in input
- Rounding errors can rapidly change the output
- Consider $f(\boldsymbol{x}) = A^{-1}\boldsymbol{x}$
  - $A \in R^{n \times n}$ has a eigendecomposition
    - Its condition no. is $\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$, i.e. ratio of largest to smallest eigenvalue
    - When this large, the output is very sensitive to input error
    - Poorly conditioned matrices amplify pre-existing errors when we multiply by its inverse

Gradient-Based Optimization

- Most ML algorithms involve optimization
- Minimize/maximize a function $f(\boldsymbol{x})$ by altering $\boldsymbol{x}$
  - Usually stated a minimization
  - Maximization accomplished by minimizing $-f(\boldsymbol{x})$
- $f(\boldsymbol{x})$ referred to as objective function or criterion
  - In minimization also referred to as loss function cost, or error
  - Example is linear least squares $\boxed{f(x) = \frac{1}{2} || Ax - b ||^2}$
  - Denote optimum value by $\boldsymbol{x}^* = \text{argmin } f(\boldsymbol{x})$

Calculus in Optimization

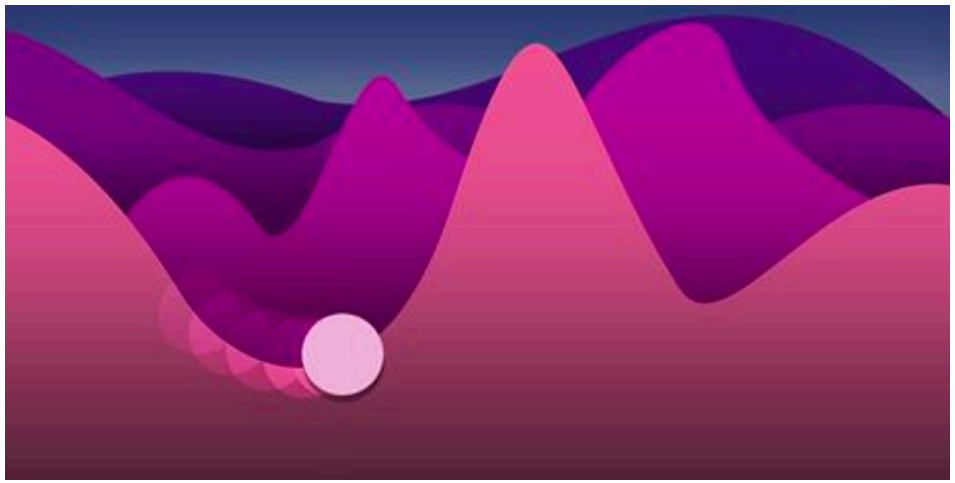
- Suppose function $y=f(x)$, x, y real nos.
  - Derivative of function denoted: $f'(x)$ or as dy/dx
    - Derivative $f'(x)$ gives the slope of $f(x)$ at point x
    - It specifies how to scale a small change in input to obtain a corresponding change in the output:

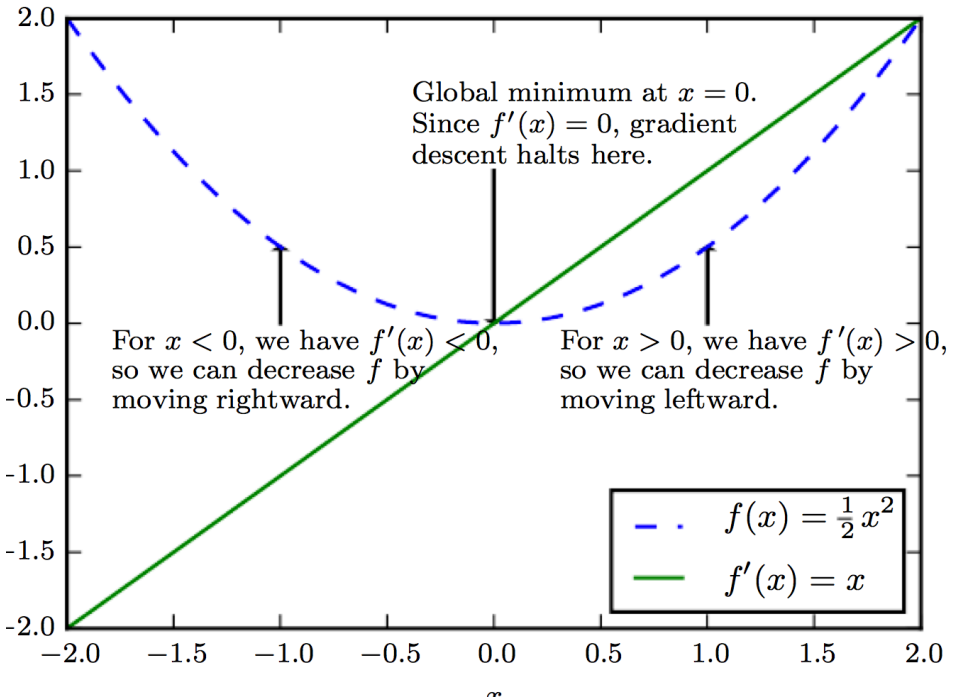
$$f(x+\varepsilon) \approx f(x) + \varepsilon f'(x)$$

  - It tells how you make a small change in input to make a small improvement in y
  - We know that $f(x - \varepsilon \operatorname{sign}(f'(x)))$ is less than $f(x)$ for small ε. Thus we can reduce $f(x)$ by moving x in small steps with opposite sign of derivative
    - This technique is called *gradient descent* (Cauchy 1847)

Gradient Descent Illustrated

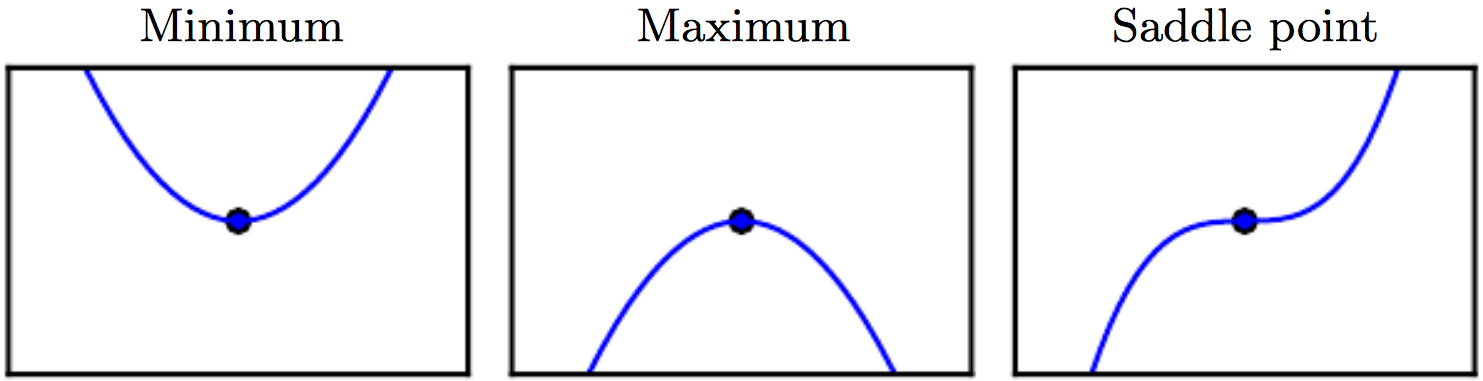
- For $x>0$, $f(x)$ increases with x and $f'(x)>0$
- For $x<0$, $f(x)$ is decreases with x and $f'(x)<0$
- Use $f'(x)$ to follow function downhill
- Reduce $f(x)$ by going in direction opposite sign of derivative $f'(x)$





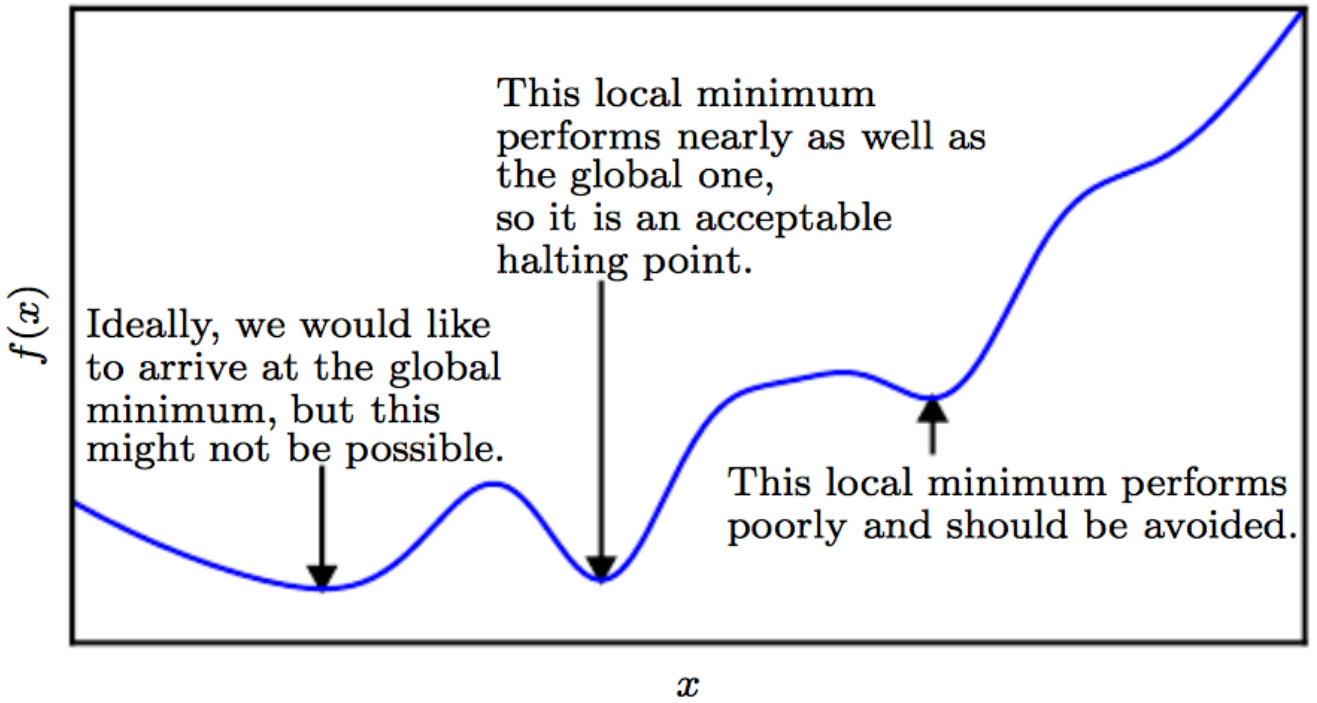
Stationary points, Local Optima

- When $f'(x)=0$ derivative provides no information about direction of move
- Points where $f'(x)=0$ are known as *stationary* or critical points
  - Local minimum/maximum: a point where $f(x)$ lower/ higher than all its neighbors
  - Saddle Points: neither maxima nor minima



Presence of multiple minima

- Optimization algorithms may fail to find global minimum
- Generally accept such solutions



Minimizing with multiple inputs

- We often minimize functions with multiple inputs: $f: R^n \rightarrow R$
- For minimization to make sense there must still be only one (scalar) output

Functions with multiple inputs

- Need partial derivatives
- $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ measures how f changes as only variable x_i increases at point $\boldsymbol{x}$
- Gradient generalizes notion of derivative where derivative is wrt a vector
- Gradient is vector containing all of the partial derivatives denoted $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$
  - Element i of the gradient is the partial derivative of f wrt x_i
  - Critical points are where every element of the gradient is equal to zero

Directional Derivative

- Directional derivative in direction $\mathbf{u}$ (a unit vector) is the slope of function f in direction $\mathbf{u}$
  - This evaluates to $\boxed{\mathbf{u}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x})}$
- To minimize f find direction in which f decreases the fastest
  - Do this using $\boxed{\min_{\mathbf{u},\mathbf{u}^T\mathbf{u}=1} \mathbf{u}^T \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \min_{\mathbf{u},\mathbf{u}^T\mathbf{u}=1} \|\mathbf{u}\|_2 \|\nabla_{\boldsymbol{x}} f(\boldsymbol{x})\|_2 \cos\theta}$
    - where θ is angle between $\mathbf{u}$ and the gradient
    - Substitute $\|\mathbf{u}\|_2=1$ and ignore factors that not depend on $\mathbf{u}$ this simplifies to $\min_u \cos\theta$
    - This is minimized when $\mathbf{u}$ points in direction opposite to gradient
    - In other words, the *gradient points directly uphill, and the negative gradient points directly downhill*

Method of Gradient Descent

- The gradient points directly uphill, and the negative gradient points directly downhill
- Thus we can decrease f by moving in the direction of the negative gradient
  - This is known as the method of steepest descent or gradient descent
- Steepest descent proposes a new point

$$\boldsymbol{x}' = \boldsymbol{x} - \varepsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

  - where ε is the learning rate, a positive scalar. Set to a small constant.

Choosing ε: Line Search

- We can choose ε in several different ways
- Popular approach: set ε to a small constant
- Another approach is called *line search*:
- Evaluate $f(\boldsymbol{x} - \varepsilon \nabla_x f(\boldsymbol{x}))$ for several values of ε and choose the one that results in smallest objective function value

Ex: Gradient Descent on Least Squares

- Criterion to minimize $f(\mathrm{x}) = \frac{1}{2} || A\mathrm{x} - \mathrm{b} ||^2$
  - Least squares regression $E_D(\mathrm{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathrm{w}^T\phi(x_n)\}^2$
- The gradient is

$$\nabla_x f(\boldsymbol{x}) = A^T(A\boldsymbol{x} - b) = A^T A\boldsymbol{x} - A^T\boldsymbol{b}$$

- Gradient Descent algorithm is
  1. Set step size ε, tolerance δ to small, positive nos.
  2. *while* $|| A^T A\boldsymbol{x} - A^T\boldsymbol{b} ||_2 > \delta$ do

     $\boldsymbol{x} \leftarrow \boldsymbol{x} - \varepsilon(A^T A\boldsymbol{x} - A^T\boldsymbol{b})$
  3. *end while*

Convergence of Steepest Descent

- Steepest descent converges when every element of the gradient is zero
  - In practice, very close to zero
- We may be able to avoid iterative algorithm and jump to the critical point by solving the equation $\nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}\right) = 0$ for $\boldsymbol{x}$

Generalization to discrete spaces

- Gradient descent is limited to continuous spaces
- Concept of repeatedly making the best small move can be generalized to discrete spaces
- Ascending an objective function of discrete parameters is called *hill climbing*

Beyond Gradient: Jacobian and Hessian matrices

- Sometimes we need to find all derivatives of a function whose input and output are both vectors
- If we have function $f: R^m \rightarrow R^n$
  - Then the matrix of partial derivatives is known as the Jacobian matrix J defined as

$$J_{i,j} = \frac{\partial}{\partial x_j} f(x)_i$$

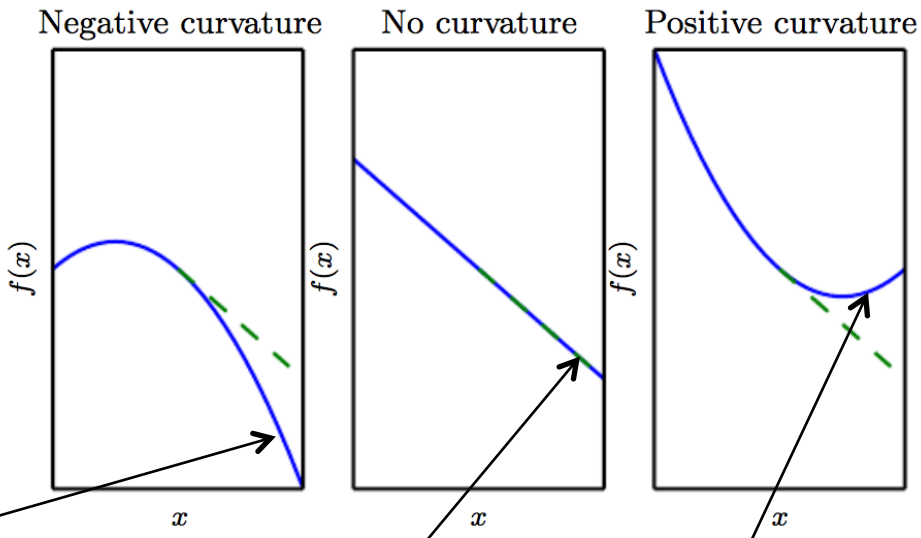
Second derivative

- Derivative of a derivative
- For a function $f: R^n \rightarrow R$ the derivative wrt x_i of the derivative of f wrt x_j is denoted as $\frac{\partial^2}{\partial x_i \partial x_j} f$
- In a single dimension we can denote $\frac{\partial^2}{\partial x^2} f$ by $f''(x)$
- Tells us how the first derivative will change as we vary the input
- This important as it tells us whether a gradient step will cause as much of an improvement as based on gradient alone

Second derivative measures curvature

- Derivative of a derivative
- Quadratic functions with different curvatures

Dashed line is value of cost function predicted by gradient alone



Decrease is faster than predicted by Gradient Descent

Gradient Predicts decrease correctly

Decrease is slower than expected Actually increases

Hessian

- Second derivative with many dimensions
- $H(f)(x)$ is defined as $$H(f)(x)_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(x)$$
- Hessian is the Jacobian of the gradient
- Hessian matrix is symmetric, i.e., $H_{i,j} = H_{j,i}$
  - anywhere that the second partial derivatives are continuous
  - So the Hessian matrix can be decomposed into a set of real eigenvalues and an orthogonal basis of eigenvectors
    - Eigenvalues of H are useful to determine learning rate as seen in next two slides

Role of eigenvalues of Hessian

- Second derivative in direction d is $\mathrm{d}^T H \mathrm{d}$
  - If d is an eigenvector, second derivative in that direction is given by its eigenvalue
  - For other directions, weighted average of eigenvalues (weights of 0 to 1, with eigenvectors with smallest angle with d receiving more value)
- Maximum eigenvalue determines maximum second derivative and minimum eigenvalue determines minimum second derivative

Learning rate from Hessian

- Taylor's series of $f(\boldsymbol{x})$ around current point $\boldsymbol{x}^{(0)}$

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^T \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^T H(\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

  - where $\boldsymbol{g}$ is the gradient and H is the Hessian at $\boldsymbol{x}^{(0)}$

– If we use learning rate ε the new point $\boldsymbol{x}$ is given by $\boldsymbol{x}^{(0)} - \varepsilon \boldsymbol{g}$. Thus we get $f(\boldsymbol{x}^{(0)} - \varepsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \varepsilon \boldsymbol{g}^T \boldsymbol{g} + \frac{1}{2}\varepsilon^2 g^T H g$

  - There are three terms:
    - original value of f,
    - expected improvement due to slope, and
    - correction to be applied due to curvature
  - Solving for step size when correction is least gives

$$\varepsilon^* \approx \frac{\boldsymbol{g}^T \boldsymbol{g}}{\boldsymbol{g}^T H \boldsymbol{g}}$$

Second Derivative Test: Critical Points

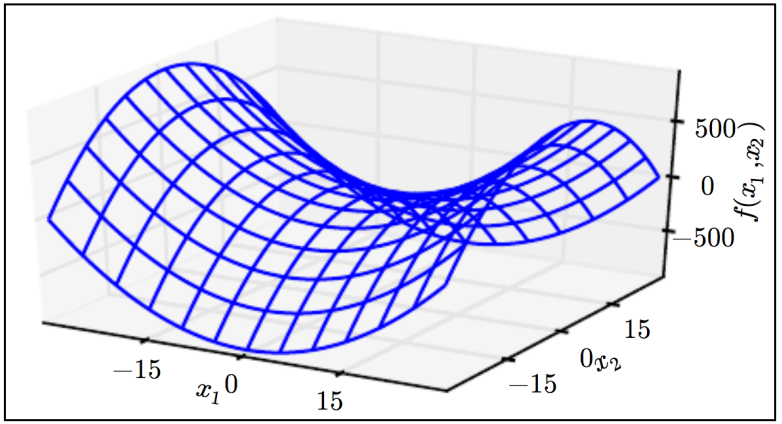
- On a critical point $f'(x)=0$
- When $f''(x)>0$ the first derivative $f'(x)$ increases as we move to the right and decreases as we move left
- We conclude that x is a local minimum
- For local maximum, $f'(x)=0$ and $f''(x)<0$
- When $f''(x)=0$ test is inconclusive: x may be a saddle point or part of a flat region

Multidimensional Second derivative test

- In multiple dimensions, we need to examine second derivatives of all dimensions
- Eigendecomposition generalizes the test
- Test eigenvalues of Hessian to determine whether critical point is a local maximum, local minimum or saddle point
- When H is positive definite (all eigenvalues are positive) the point is a local minimum
- Similarly negative definite implies a maximum

Saddle point

- Contains both positive and negative curvature
- Function is $f(\boldsymbol{x}) = x_1^2 - x_2^2$



  - Along axis x_1, function curves upwards: this axis is an eigenvector of H and has a positive value
  - Along x_2, function corves downwards; its direction is an eigenvector of H with negative eigenvalue
  - At a saddle point eigen values are both positive and negative

Inconclusive Second Derivative Test

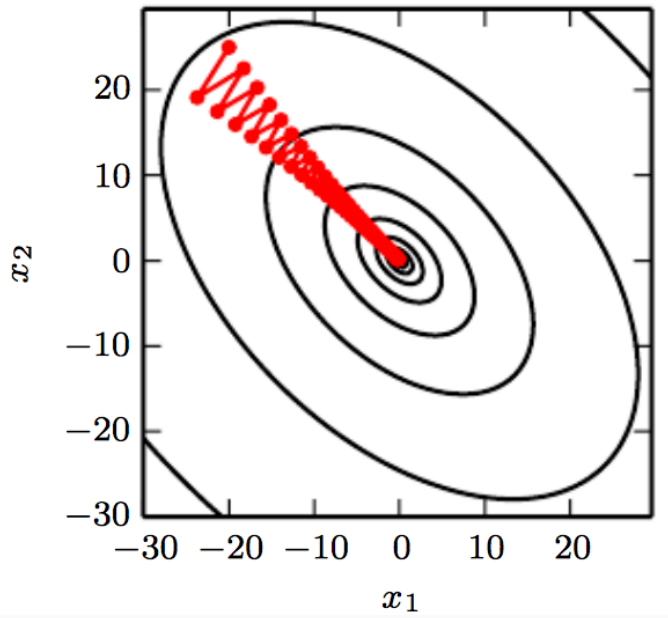
- Multidimensional second derivative test can be inconclusive just like univariate case
- Test is inconclusive when all non-zero eigen values have same sign but at least one value is zero
  - since univariate second derivative test is inconclusive in cross-section corresponding to zero eigenvalue

Poor Condition Number

- There are different second derivatives in each direction at a single point
- Condition number of H e.g., $\lambda_{\max}/\lambda_{\min}$ measures how much they differ
  - Gradient descent performs poorly when H has a poor condition no.
  - Because in one direction derivative increases rapidly while in another direction it increases slowly
  - Step size must be small so as to avoid overshooting the minimum, but it will be too small to make progress in other directions with less curvature

Gradient Descent without H

- H with condition no, 5
  - Direction of most curvature has five times more curvature than direction of least curvature
- Due to small step size Gradient descent wastes time
- Algorithm based on Hessian can predict that steepest descent is not promising



Newton's method uses Hessian

- Another second derivative method
  - Using Taylor's series of $f(\boldsymbol{x})$ around current $\boldsymbol{x}^{(0)}$

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^T \nabla_x f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^T H(f)(\boldsymbol{x} - \boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)})$$

    - solve for the critical point of this function to give

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - H(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_x f(\boldsymbol{x}^{(0)})$$

  - When f is a quadratic (positive definite) function use solution to jump to the minimum function directly
  - When not quadratic apply solution iteratively
- Can reach critical point much faster than gradient descent
  - But useful only when nearby point is a minimum

Summary of Gradient Methods

- First order optimization algorithms: those that use only the gradient
- Second order optimization algorithms: use the Hessian matrix such as Newton's method
- Family of functions used in ML is complicated, so optimization is more complex than in other fields
  - No guarantees
- Some guarantees by using Lipschitz continuous functions, $\left|f(\boldsymbol{x})-f(\boldsymbol{y})\right| \leq L\left\|\boldsymbol{x}-\boldsymbol{y}\right\|_2$
  - with Lipschitz constant L

Convex Optimization

- Applicable only to convex functions– functions which are well-behaved,
  - e.g., lack saddle points and all local minima are global minima
- For such functions, Hessian is positive semi-definite everywhere
- Many ML optimization problems, particularly deep learning, cannot be expressed as convex optimization

Constrained Optimization

- We may wish to optimize $f(\boldsymbol{x})$ when the solution $\boldsymbol{x}$ is constrained to lie in set S
  - Such values of $\boldsymbol{x}$ are feasible solutions
- Often we want a solution that is small, such as $||\boldsymbol{x}|| \leq 1$
- Simple approach: modify gradient descent taking constraint into account (using Lagrangian formulation)

Ex: Least squares with Lagrangian

- We wish to minimize $f(\boldsymbol{x}) = \frac{1}{2} || A\boldsymbol{x} - \mathrm{b} ||^2$
- Subject to constraint $\boldsymbol{x}^{\mathrm{T}}\boldsymbol{x} \leq 1$
- We introduce the Lagrangian $L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda\left(\boldsymbol{x}^T\boldsymbol{x} - 1\right)$
  - And solve the problem $\min_{\boldsymbol{x}} \max_{\lambda, \lambda \geq 0} L(\boldsymbol{x}, \lambda)$
- For the unconstrained problem (no Lagrangian) the smallest norm solution is $\boldsymbol{x} = \mathrm{A}^{+}\mathrm{b}$
  - If this solution is not feasible, differentiate Lagrangian wrt x to obtain $A^{\mathrm{T}}A\boldsymbol{x} - A^{\mathrm{T}}\mathrm{b} + 2\lambda\,\boldsymbol{x} = 0$
  - Solution takes the form $\boldsymbol{x} = (A^{\mathrm{T}}A + 2\lambda\, I)^{-1}A^{T}b$
  - Choosing λ: continue solving linear equation and increasing λ until $\boldsymbol{x}$ has the correct norm

Generalized Lagrangian: KKT

- More sophisticated than Lagrangian
- Karush-Kuhn-Tucker is a very general solution to constrained optimization
- While Lagrangian allows equality constraints, KKT allows both equality and inequality constraints
- To define a generalized Lagrangian we need to describe S in terms of equalities and inequalities

Generalized Lagrangian

- Set S is described in terms of m functions $g(i)$ and n functions $h(j)$ so that

$$S = \left\{ \boldsymbol{x} \mid \forall i, g^{(i)}(\boldsymbol{x}) = 0 \text{ and } \forall j, h^{(j)}(\boldsymbol{x}) \leq 0 \right\}$$

  - Functions of g are equality constraints and functions of h are inequality constraints
- Introduce new variables λ_i and α_j for each constraint (called KKT multipliers) giving the generalized Lagrangian

$$\boxed{L(\boldsymbol{x}, \lambda, \alpha) = f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x})}$$

- We can now solve the unconstrained optimization problem

Gradient

- Essential role of calculus