Deep learning book Ch. 4- Numerical computation notes

James Chuang

February 3, 2017

Contents

My notes on Deep Learning Book Chapter 4 on Numerical Computation.

ML algorithms usually require a high amount of numerical computation, which typically refers to algorithms that solve mathematical problems by methods that update estimates of a solution iteratively, rather than solving analytically. Evaluating a function involving real numbers on a computer can be difficult, because the real numbers must be represented with a finite amount of memory.

DL 4.1 overflow and underflow

The fundamental difficulty in performing continuous math on a digital computer is the need to represent infinitely many real numbers with a finite number of bit patterns. This leads to approximation error, which can just be rounding error. Rounding error is especially problematic when it compounds across many operations, and can cause algorithms to fail.

Forms of rounding error:

- ***underflow***: when numbers near zero are rounded to zero. This is especially a problem when it leads to dividing by zero or taking the log of zero.
- ***overflow***: when numbers with large magnitude are approximated as ∞ or $-\infty$.

The softmax function as a function that must be stabilized against underflow and overflow.

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

Analytically, when all of the x_i are equal to a constant c, all of the outputs should be $\frac{1}{n}$. Numerically, this may not occur when c has large magnitude. For very negative c, $\exp(c)$ will underflow, causing the denominator to become 0, and making the output undefined. For very positive c, $\exp(c)$ will overflow, again making the output undefined. Both of these difficulties can be resolved by instead evaluating softmax(Z), where $Z = X - \max_i x_i$. The value of the softmax function is not changed analytically by adding or subtracting a scalar from the input vector:

$$\begin{aligned}
\text{softmax}(Z)_i &= \frac{\exp(x_i - \max_i x_i)}{\sum_{j=1}^{n} \exp(x_j - \max_i x_i)} \\
&= \frac{\frac{\exp(x_i)}{\exp(\max_i x_i)}}{\sum_{j=1}^{n} \frac{\exp(x_j)}{\exp(\max_i x_i)}} \\
&= \frac{\frac{\exp(x_i)}{\exp(\max_i x_i)}}{\frac{\sum_{j=1}^{n} \exp(x_j)}{\exp(\max_i x_i)}} \\
&= \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} \\
\text{softmax}(Z)_i &= \text{softmax}(X)_i
\end{aligned}$$

Subtracting $\max_i x_i$ makes the largest argument to $\exp$ be 0, which protects against overflow. Likewise, at least one term in the denominator has a value of 1, protecting against underflow in the denominator and subsequent division by zero. However, underflow in the numerator can still cause the expression to erroneously evaluate to zero. Then, taking $\log$ softmax(x) would erroneously return $-\infty$. This can be stabilized using the same trick used to stabilize the softmax function above.

DL 4.2 poor conditioning

Conditioning: how rapidly a function changes with respect to small changes in its inputs.

Functions that change rapidly in response to small input perturbations can be problematic because rounding error in the inputs can result in large changes in the output.

Consider the function $f(x) = \mathbf{A}^{-1}x$. When $\mathbf{A} \in \mathbb{R}^{n \times n}$ has an eigenvalue decomposition, its ***condition number*** is

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|$$

, the ratio of the largest and smallest eigenvalue. When the condition number is large, matrix inversion is particularly sensitive to error in the input.

DL 4.3 gradient-based optimization

Optimization: minimizing a function $f(x)$ by altering x. (Maximization is simply minimizing $-f(x)$).

- ***objective function*** or ***criterion***: the function to be minimized or maximized
  - When the function is to be minimized, it can also be called the ***cost function***, ***loss function***, or ***error function***.

The value that minimizes or maximizes a function can be denoted with a superscript $*$, i.e. $x^* = \arg\min f(x)$.

The derivative of a function is useful for function minimization because it tells us how to change x in order to make a small improvement in y. For small enough ϵ, $f(x - \epsilon \text{ sign}(f'(x)))$ is less than $f(x)$. This technique is called ***gradient descent***.

- ***critical points*** or ***stationary points***: points where $f'(x) = 0$
  - can be a ***local minimum***, ***local maximum***, or ***saddle point***
- ***global minimum***: The point with the absolute lowest value of $f(x)$ (there can be more than one).
  - In deep learning, the functions may have many suboptimal local minima, and many saddle points surrounded by flat regions. This makes optimization difficult, especially for high-dimensional functions. We therefore usually settle for finding a "very low" value of f, which is not necessarily a global minimum.

The above can be generalized to functions with multiple inputs $f : \mathbb{R}^n \to \mathbb{R}$. (In order to minimize the function, the output must still be a single scalar output). In this case, the ***gradient*** $\nabla_x f(x)$ is analogous to the derivative. The ***directional derivative*** in a direction u (a unit vector) is the slope of the function f in direction u, i.e. $\frac{\partial}{\partial \alpha} f(x + \alpha u) = u^T \nabla_x f(x)$. The function f can be minimized by moving in the direction of the negative gradient (This is known as the ***method of steepest descent*** or ***gradient descent***):

$$x' = x - \epsilon \nabla_x f(x)$$

, where ϵ is the ***learning rate***, a positive scalar determining the step size.

How to set the learning rate ϵ?

- popular approach is to set ϵ to a small constant
- or, solve for step size that makes directional derivative zero
- or, ***line search***: evaluate $f(\mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}))$ for several values of ϵ and choose the one that results in the smallest objective function value

Steepest descent converges when $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$

- in cases where there is an analytical solution, this point can be jumped to directly

Gradient descent is limited to optimization in continuous spaces

- the concept of making small moves towards better configurations can be generalized to discrete spaces (this is called ***hill climbing***)

DL 4.3.1 Jacobian and Hessian Matrices

Sometimes need the partial derivatives of a function $f : \mathbb{R}^m \to \mathbb{R}^n$.

Jacobian matrix: $\mathbf{J}(f(\mathbf{x}))$ the matrix of all partial derivatives

$$J \in \mathbb{R}^{n \times m}, \quad J_{i,j} = \frac{\partial}{\partial x_j} f(x)_i$$

second derivatives indicate *curvature*

- important because it tells us whether a gradient step will cause as much of an improvement as we would expect based on the gradient alone
- $f''(x) = 0$: no curvature, value can be predicted using only the gradient
- $f''(x) < 0$: function curves downward, so cost function decreases by more than ϵ
- $f''(x) > 0$: function curves upward, so cost function decreases by less than ϵ

Hessian matrix: $\mathbf{H}(f(\mathbf{x}))$ the matrix of all second derivatives

$$H(f(\mathbf{x}))_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x})$$

Anywhere that the second partial derivatives are continuous, $\frac{\partial^2}{\partial x_i \partial x_j} f(\mathbf{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\mathbf{x})$, implying that $H_{i,j} = H_{j,i}$, meaning that the Hessian is symmetric at such points.

- Most of the functions we will encounter have symmetric Hessian almost everywhere
- because the Hessian matrix is real and symmetric, it can be decomposed into a set of real eigenvalues and an orthogonal basis of eigenvectors
  - the second derivative in the direction of a unit vector $\mathbf{d}$ is given by $\mathbf{d}^T \mathbf{H} \mathbf{d}$
    - when $\mathbf{d}$ is an eigenvector of $\mathbf{H}$, the second derivative in the direction of $\mathbf{d}$ is given by the corresponding eigenvalue
    - for other directions, the second derivative is a weighted average of all of the eigenvalues, with weights between 0 and 1
    - max eigenvalue $\to$ max second derivative
    - min eigenvalue $\to$ min second derivative

The directional second derivative measures how well a gradient descent step is expected to perform. The second order Taylor series approximation of the function $f(\mathbf{x})$ about the current point $\mathbf{x}^{(0)}$:

$$f(\mathbf{x}) \approx f\left(\mathbf{x}^{(0)}\right) + \left(\mathbf{x} - \mathbf{x}^{(0)}\right)^T \mathbf{g} + \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^{(0)}\right)^T \mathbf{H}\left(\mathbf{x} - \mathbf{x}^{(0)}\right) \qquad \mathbf{g} = \nabla_x \left(f\left(\mathbf{x}^{(0)}\right)\right), \ \mathbf{H} = \mathbf{H}\left(f\left(\mathbf{x}^{(0)}\right)\right)$$

Using gradient descent, the new point $\mathbf{x} = \mathbf{x}^{(0)} - \epsilon \nabla_x \left(f(\mathbf{x}^{(0)})\right)$. Substituting this in:

$$\begin{aligned}
f(\mathbf{x}) &\approx f\left(\mathbf{x}^{(0)}\right) + \left(\mathbf{x} - \mathbf{x}^{(0)}\right)^T \mathbf{g} + \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^{(0)}\right)^T \mathbf{H}\left(\mathbf{x} - \mathbf{x}^{(0)}\right) \\
f\left(\mathbf{x}^{(0)} - \epsilon \mathbf{g}\right) &\approx f\left(\mathbf{x}^{(0)}\right) + \left(\mathbf{x}^{(0)} - \epsilon \mathbf{g} - \mathbf{x}^{(0)}\right)^T \mathbf{g} + \frac{1}{2}\left(\mathbf{x}^{(0)} - \epsilon \mathbf{g} - \mathbf{x}^{(0)}\right)^T \mathbf{H}\left(\mathbf{x}^{(0)} - \epsilon \mathbf{g} - \mathbf{x}^{(0)}\right) \\
f\left(\mathbf{x}^{(0)} - \epsilon \mathbf{g}\right) &\approx f\left(\mathbf{x}^{(0)}\right) - \epsilon \mathbf{g}^T \mathbf{g} + \frac{1}{2}\epsilon^2 \mathbf{g}^T \mathbf{H} \mathbf{g}
\end{aligned}$$

new value of $f \approx$ original value $-$ expected improvement due to slope $+$ correction for curvature

- $\mathbf{g}^T\mathbf{H}\mathbf{g}$ very positive $\rightarrow$ gradient descent step can actually move uphill
- $\mathbf{g}^T\mathbf{H}\mathbf{g}$ zero or negative $\rightarrow$ Taylor series approx. predicts that $\epsilon \rightarrow \infty$ will decrease f infinitely
  - in practice, must then resort to heuristic choices of ϵ
- $\mathbf{g}^T\mathbf{H}\mathbf{g}$ positive, then solving for the optimal step size:

$$\epsilon^* = \frac{\mathbf{g}^T\mathbf{g}}{\mathbf{g}^T\mathbf{H}\mathbf{g}}$$

  - worst case: $\mathbf{g}$ aligns with the eigenvector of $\mathbf{H}$ corresponding to the maximal eigenvalue λ_{max}
    - then, optimal step size $= \frac{1}{\lambda_{\text{max}}}$
    - thus, to the extent that the function we minimize can be approximated well by a quadratic function, the eigenvalues of the Hessian determine the scale of the learning rate
- ***second derivative test***: determine whether a critical point is a local maximum, local minimum, or saddle point
  - at critical points, $f'(x) = 0$
    - if $f''(x) > 0$, $f'(x)$ increases to the right, and $f'(x)$ decreases to the left
      - i.e., $f'(x - \epsilon) < 0$ and $f'(x + \epsilon) > 0$ for small ϵ
      - therefore, x is a local minimum
    - similarly, if $f' = 0$ and $f''(x) < 0$, x is a local maximum
    - if $f'' = 0$, test is inconclusive
      - x may be a saddle point, or part of a flat region
- using the eigendecomposition of the Hessian matrix, the second derivative test can be generalized to multiple dimensions
  - at critical points, $\nabla_{\mathbf{x}} f(\mathbf{x}) = 0$
    - if Hessian is **positive definite** $\rightarrow$ all eigenvalues positive $\rightarrow$ directional second derivative in any direction is positive $\rightarrow x$ is a **local minimum**
    - if Hessian is **negative definite** $\rightarrow$ all eigenvalues negative $\rightarrow$ directional second derivative in any direction is negative $\rightarrow x$ is a **local maximum**
    - if Hessian has **at least one positive eigenvalue and one negative eigenvalue**, x is a local max in at least one dimension and a local min in at least one dimension, and therefore is a **saddle point**
    - if **all non-zero eigenvalues have the same sign, but at least one eigenvalue is zero, the test is inconclusive** because the univariate second derivative test is inconclusive in the cross section corresponding to the zero eigenvalue
- in multiple dimensions, there is a different second derivative for each direction
  - the ***condition number*** of the Hessian measures how much the second derivatives vary
    - poor condition number means gradient descent performs poorly, because gradient descent is unaware that it should preferentially explore only in the steep directions
      - this also makes it difficult to choose a step size:
        - step size must be small to avoid overshooting the minimum and going uphill in directions with strong positive curvature
        - this small step size is usually too small to make significant progress in directions with little curvature
        - this can be resolved by using the Hessian to guide the search, e.g. with the ***Newton-Raphson method***

The Newton-Raphson method is based on using the second-order Taylor series expansion to approximate $f(\mathbf{x})$ near some point $\mathbf{x}^{(0)}$

$$f(\mathbf{x}) \approx f\left(\mathbf{x}^{(0)}\right) + \left(\mathbf{x} - \mathbf{x}^{(0)}\right)^T \nabla_{\mathbf{x}} f\left(\mathbf{x}^{(0)}\right) + \frac{1}{2}\left(\mathbf{x} - \mathbf{x}^{(0)}\right)^T \mathbf{H} f\left(\mathbf{x}^0\right)\left(\mathbf{x} - \mathbf{x}^{(0)}\right)$$

Solving for the critical point:

$$\mathbf{x}^* = \mathbf{x}^{(0)} - \mathbf{H} f\left(\mathbf{x}^{(0)}\right)^{-1} \nabla_{\mathbf{x}} f\left(\mathbf{x}^{(0)}\right)$$

- For a positive definite quadratic function, Newton's method arrives at the minimum in one step
- if f is not truly quadratic but can be locally approximated as positive definite quadratic, Newton's method is iteratively applied
  - this can arrive at the minimum much faster than gradient descent
    - this is useful near a local minimum, but can be harmful near a saddle point
    - Newton's method is only appropriate when the nearby critical point is a minimum
      - in contrast, gradient descent is not attracted to saddle points unless the gradient points toward them
- gradient descent is a ***first order optimization algorithm***

- methods such as the Newton-Raphson method that take the Hessian into account are called ***second order optimization algorithms***

Optimization algorithms come with almost no guarantees, because the family of functions encountered is very complicated

- in deep learning, can sometimes gain some guarantees by restricting to functions that are ***Lipschitz continuous*** or have Lipschitz continuous derivatives
  - a Lipschitz continuous function is a function f whose rate of change is bounded by a ***Lipschitz constant*** $\mathcal{L}$:

$$\forall \mathbf{x}, \mathbf{y}, \quad |f(\mathbf{x}) - f(\mathbf{y})| \leq \mathcal{L}\|\mathbf{x} - \mathbf{y}\|_2$$

  - this allows us to quantify the assumption that a small change in the input made by an algorithm such as gradient descent will have a small change in the output
    - Lipschitz continuity is also a fairly weak constraint
    - many optimization problems in deep learning can be made Lipschitz continuous with relatively minor modifications

The most successful field of specialized optimization is ***convex optimization***

- convex optimization algorithms are able to provide more guarantees by making stronger restrictions
  - applicable only to convex functions, i.e. functions for which the Hessian is positive semidefinite everywhere
    - these functions are well-behaved:
      - they lack saddle points
      - all local minima are necessarily global minima
  - most problems in deep learning are difficult to express in terms of convex optimization
    - used only as a subroutine of some DL algorithms

DL 4.4 Constrained Optimization

- ***constrained optimization***: optimizing $f(\mathbf{x})$ for values of x in some set $\mathbb{S}$
  - ***feasible points***: points x in the set $\mathbb{S}$
- often wish to find a solution that is "small" in some sense
  - common solution is a norm constraint, e.g. $\|\mathbf{x}\| \leq 1$
- the ***Karush-Kuhn-Tucker*** (KKT) approach: a very general solution to constrained optimization (also see CS229 notes 3)
  - design a different, unconstrained optimization problem whose solution can be converted into a solution to the original, constrained optimization problem
  - do this by introducing a new function- the ***generalized Lagrangian*** or ***generalized Lagrange function***
    - first, describe $\mathbb{S}$ in terms of m functions $g^{(i)}$ and n functions $h^{(j)}$ such that

$$\mathbb{S} = \left\{ \mathbf{x} \mid \begin{array}{ll} \forall i, g^{(i)}(\mathbf{x}) = 0 & \text{equality constraints} \\ \forall j, h^{(j)}(\mathbf{x}) \leq 0 & \text{inequality constraints} \end{array} \right\}$$

    - introduce KKT multipliers λ_i and α_j for each constraint, then define the generalized Lagrangian:

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

    - consider the following quantity:

$$\max_{\lambda, \alpha, \alpha \geq 0} L(\mathbf{x}, \lambda, \alpha).$$

      Any time a constraint is violated ($g^{(i)} \neq 0$ or $h^{(i)} > 0$),

$$\max_{\lambda, \alpha, \alpha \geq 0} L(\mathbf{x}, \lambda, \alpha) = \infty$$

      while when the constraints are all satisfied,

$$\max_{\lambda, \alpha, \alpha \geq 0} L(\mathbf{x}, \lambda, \alpha) = f(x).$$

This means that

$$\min_{\mathbf{x}} \max_{\lambda,\alpha,\alpha\geq 0} L(\mathbf{x},\lambda,\alpha) = \min_{\mathbf{x}\in\mathbb{S}} f(\mathbf{x}),$$

i.e., the solution to the new, unconstrained problem is the same as the original constrained problem.

- regarding the inequality constraints:
  - a constraint $h^{(i)}(\mathbf{x}) \leq 0$ is **active** if it holds with equality, i.e. $h^{(i)}(\mathbf{x}) = 0$
    - if a constraint is not active, then the solution to the problem using that constraint would remain at least a local solution if that constraint were removed
    - because an inactive $h^{(i)}$ has a negative value, the solution to

$$\min_{\mathbf{x}} \max_{\lambda,\alpha,\alpha\geq 0} L(\mathbf{x},\lambda,\alpha)$$

will have $\alpha_i = 0$

    - thus, $\alpha\mathbf{h}(\mathbf{x}) = 0$
    - i.e., at least one of the constraints $\alpha_i \geq 0$ or $h^{(i)} \leq 0$ must be active at the solution
      - the solution is either on the boundary imposed by the inequality and its KKT multiplier influences the solution, or the inequality has no influence on the solution and its KKT multiplier is zero
- the above properties are the ***Karush-Kuhn-Tucker (KKT) condtions***, i.e.:
  - the gradient of the generalized Lagrangian is zero
  - all constraints on both x and the KKT multipliers are satisfied
  - $\alpha \odot \mathbf{h}(\mathbf{x}) = \mathbf{0}$

DL 4.5 Example: Linear Least Squares

Minimize the squared error loss function

$$f(\mathbf{x}) = \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$$

This can be analytically solved by taking the gradient and setting it to zero, since it is a positive definite quadratic function. However, it can also be used as a simple test case for gradient-based optimization.

Finding the gradient:

$$\begin{aligned}
f(\mathbf{x}) &= \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \\
&= \frac{1}{2}\left(\mathbf{A}\mathbf{x} - \mathbf{b}\right)^T\left(\mathbf{A}\mathbf{x} - \mathbf{b}\right) \\
&= \frac{1}{2}\left[\left(\mathbf{A}\mathbf{x}\right)^T\left(\mathbf{A}\mathbf{x}\right) - \left(\mathbf{A}\mathbf{x}\right)^T\mathbf{b} - \mathbf{b}^T\left(\mathbf{A}\mathbf{x}\right) + \mathbf{b}^T\mathbf{b}\right] \\
&= \frac{1}{2}\left[\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{b}^T\mathbf{b}\right] \\
\nabla_{\mathbf{x}} f(\mathbf{x}) &= \frac{1}{2}\left[2\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{A}^T\mathbf{b}\right] \\
&= \mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}
\end{aligned}$$

Pseudocode for minimization by gradient descent:

- initialize step size ϵ and convergence tolerance δ to small, positive numbers
- **while** $\|\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}\|_2 > \delta$ **do**
  - $\mathbf{x} \leftarrow \mathbf{x} - \epsilon\left(\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b}\right)$

If solving by Newton's method, the quadratic approximation is exact (since f is a positive definite quadratic function), so the minimum is reached in one step.

Suppose we want to minimize the same function, but subject to the constraint $\mathbf{x}^T\mathbf{x} \leq 1$. This constraint can be rewritten as $\mathbf{x}^T\mathbf{x} - 1 \leq 0$. Therefore, the Lagrangian is

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) + \lambda\left(\mathbf{x}^T\mathbf{x} - 1\right),$$

and the solution to the original (primal) problem can be found by solving

$$\min_{\mathbf{x}} \max_{\lambda, \lambda \geq 0} L(\mathbf{x}, \lambda)$$

The smallest-norm solution to this unconstrained least squares problem may be found using the Moore-Penrose pseudoinverse $\mathbf{x} = \mathbf{A}^{+}\mathbf{b}$. If this point is feasible, then it is the solution to the constrained problem. Otherwise, we must find a solution where the constraint is active. Start by differentiating the Lagrangian w.r.t. $\mathbf{x}$ and setting it equal to the zero vector:

$$\begin{aligned}
L(\mathbf{x}, \lambda) &= f(\mathbf{x}) + \lambda \left(\mathbf{x}^T\mathbf{x} - 1\right) \\
&= \frac{1}{2}\left[\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{b}^T\mathbf{b}\right] + \lambda\mathbf{x}^T\mathbf{x} - \lambda \\
\nabla_{\mathbf{x}} L(\mathbf{x}, \lambda) &= \frac{1}{2}\left[2\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{A}^T\mathbf{b}\right] + \lambda\mathbf{x} = 0 \\
\mathbf{A}^T\mathbf{A}\mathbf{x} - \mathbf{A}^T\mathbf{b} + 2\lambda\mathbf{x} &= 0 \\
\left(\mathbf{A}^T\mathbf{x} + 2\lambda\mathbf{I}\right)\mathbf{x} &= \mathbf{A}^T\mathbf{b} \\
\mathbf{x} &= \left(\mathbf{A}^T\mathbf{x} + 2\lambda\mathbf{I}\right)^{-1}\mathbf{A}^T\mathbf{b}
\end{aligned}$$

The solution must take the above form, and the magnitude of λ must be chosen such that the result obeys the constraint (i.e., $\mathbf{x}^T\mathbf{x} \leq 1$). Value can be found by performing gradient ascent on λ:

$$\begin{aligned}
L(\mathbf{x}, \lambda) &= \frac{1}{2}\left[\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{x}^T\mathbf{A}^T\mathbf{b} + \mathbf{b}^T\mathbf{b}\right] + \lambda\mathbf{x}^T\mathbf{x} - \lambda \\
\frac{\partial}{\partial\lambda} L\left(\mathbf{x}, \lambda\right) &= \mathbf{x}^T\mathbf{x} - 1
\end{aligned}$$

$$\text{if} \quad \mathbf{x}^T\mathbf{x} > 1 \quad \rightarrow \quad \frac{\partial L}{\partial \lambda} > 0 \quad \rightarrow \text{gradient ascent increases } \lambda$$

An increased λ means the weight of the $\mathbf{x}^T\mathbf{x}$ penalty in the Lagrangian has increased. Therefore, solving for $\mathbf{x}$ will then yield a solution with a smaller norm. The process of solving the linear equation and adjusting λ continues until $\mathbf{x}$ has the correct norm and the derivative on λ is 0.