Deep Learning

Numerical Computations

Numerical Computation

- Algorithms that solve mathematical problems by methods that update estimates of the solution via an iterative process, rather than analytically deriving a formula providing a symbolic expression for the correct solution.
- Common operations include optimization (finding the value of an argument that minimizes or maximizes a function) and solving systems of linear equations.
- Even just evaluating a mathematical function on a digital computer can be difficult when the function involves real numbers, which cannot be represented precisely using a finite amount of memory.

Overflow and Underflow

- The fundamental difficulty in performing continuous math on a digital computer is that we need to represent infinitely many real numbers with a finite number of bit patterns.
- This means that for almost all real numbers, we incur some approximation error when we represent the number in the computer.
- In many cases, this is just rounding error.
- Rounding error is problematic, especially when it compounds across many operations, and can cause algorithms that work in theory to fail in practice if they are not designed to minimize the accumulation of rounding error.

Overflow and Underflow

- One form of rounding error that is particularly devastating is **underflow**.
- Underflow occurs when numbers near zero are rounded to zero.
- Many functions behave qualitatively differently when their argument is zero rather than a small positive number.
  - For example, we usually want to avoid division by zero (some software environments will raise exceptions when this occurs, others will return a result with a placeholder not-a-number value) or taking the logarithm of zero (this is usually treated as $-\infty$, which then becomes not-a-number if it is used for many further arithmetic operations).

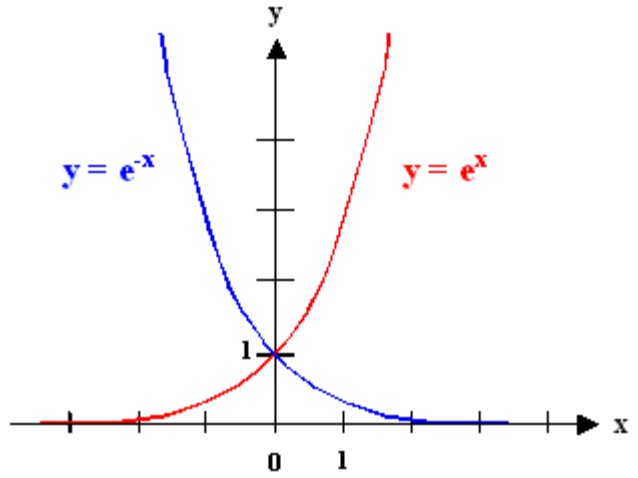
Overflow and Underflow

- Another highly damaging form of numerical error is **overflow**.
- Overflow occurs when numbers with large magnitude are approximated as ∞ or $-\infty$.
- Further arithmetic will usually change these infinite values into not-a-number values.

Overflow and Underflow

- One example of a function that must be stabilized against underflow and overflow is the softmax function.
- The softmax function is often used to predict the probabilities associated with a multinoulli distribution.
- The softmax function is defined to be

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



Overflow and Underflow

- Consider what happens when all of the x_i are equal to some constant c.
- Analytically, we can see that all of the outputs should be equal to $1/n$.
- Numerically, this may not occur when c has large magnitude.
- If c is very negative, then $\exp(c)$ will underflow.
- This means the denominator of the softmax will become 0, so the final result is undefined. When c is very large and positive, $\exp(c)$ will overflow, again resulting in the expression as a whole being undefined.
- Both of these difficulties can be resolved by instead evaluating softmax(z) where $z = x - \max_i x_i$.

Overflow and Underflow

- Simple algebra shows that the value of the softmax function is not changed analytically by adding or subtracting a scalar from the input vector.
- Subtracting $\max_i x_i$ results in the largest argument to exp being 0, which rules out the possibility of overflow.
- Likewise, at least one term in the denominator has a value of 1, which rules out the possibility of underflow in the denominator leading to a division by zero.

Overflow and Underflow

- Underflow in the numerator can still cause the expression as a whole to evaluate to zero.
- This means that if we implement log softmax(x) by first running the softmax subroutine then passing the result to the log function, we could erroneously obtain $-\infty$.
- We must implement a separate function that calculates log softmax in a numerically stable way.
- The log softmax function can be stabilized using the same trick as we used to stabilize the softmax function.

Poor Conditioning

- Conditioning refers to how rapidly a function changes with respect to small changes in its inputs.
- Functions that change rapidly when their inputs are perturbed slightly can be problematic for scientific computation because rounding errors in the inputs can result in large changes in the output.

Poor Conditioning

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

$$\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|. \tag{4.2}$$

$$\kappa(A) = \|A^{-1}\|_2 \|A\|_2$$

$$\|A\|_2 = \sigma_{max}(A)$$

$$\|A^{-1}\|_2 = \frac{1}{\sigma_{min}(A)}$$

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

- If the condition number $\kappa(A)=10^k$, then you may lose up to k digits of accuracy on top of what would be lost to the numerical method due to loss of precision from arithmetic methods.

$$\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2}$$

$$\Rightarrow \|A\|_2^2 = \max_{x \neq 0} \frac{\|Ax\|_2^2}{\|x\|_2^2}$$

$$\because \|Ax\|_2^2 = (Ax)^T Ax = x^T A^T A x$$

and $\|x\|_2^2 = x^T x$

$$\Rightarrow \|A\|_2^2 = \max_{x \neq 0} \frac{x^T A^T A x}{x^T x}$$

The quantity $\frac{x^T A^T A x}{x^T x}$ is the Rayleigh quotient of $\|A\|_2^2$ i.e. $A^T A$

The max value of Rayleigh quotient of $A^T A$ is max Eigen Value λ_{max} of $(A^T A)$

$$\|A\|_2^2 = \lambda_{max}(A^T A) \Rightarrow \|A\|_2 = \sqrt{\lambda_{max}(A^T A)}$$

Poor Conditioning

- This sensitivity is an intrinsic property of the matrix itself, not the result of rounding error during matrix inversion.
- Poorly conditioned matrices amplify pre-existing errors when we multiply by the true matrix inverse.

Gradient-Based Optimization

- Most deep learning algorithms involve optimization of some sort.
- Optimization refers to the task of either minimizing or maximizing some function $f(x)$ by altering x.
- We usually phrase most optimization problems in terms of minimizing $f(x)$.
- Maximization may be accomplished via a minimization algorithm by minimizing $-f(x)$.

Gradient-Based Optimization

- The function we want to minimize or maximize is called the **objective function** or **criterion**.
- When we are minimizing it, we may also call it the **cost function**, **loss function**, or **error function**.
- We often denote the value that minimizes or maximizes a function with a superscript $*$. For example, we might say $x^* = \text{argmin } f(x)$.

Gradient-Based Optimization

- Suppose we have a function $y = f(x)$, where both x and y are real numbers.
- The **derivative** of this function is denoted as $f'(x)$ or as dy/dx .
- The derivative $f'(x)$ gives the slope of $f(x)$ at the point x.
- it specifies how to scale a small change in the input in order to obtain the corresponding change in the output:

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

Gradient-Based Optimization

- The derivative is therefore useful for minimizing a function because it tells us how to change x in order to make a small improvement in y.
- For example, we know that $f(x - \varepsilon * \text{sign}(f'(x)))$ is less than $f(x)$ for small enough ε.
- We can thus reduce $f(x)$ by moving x in small steps with opposite sign of the derivative.
- This technique is called **gradient descent.**

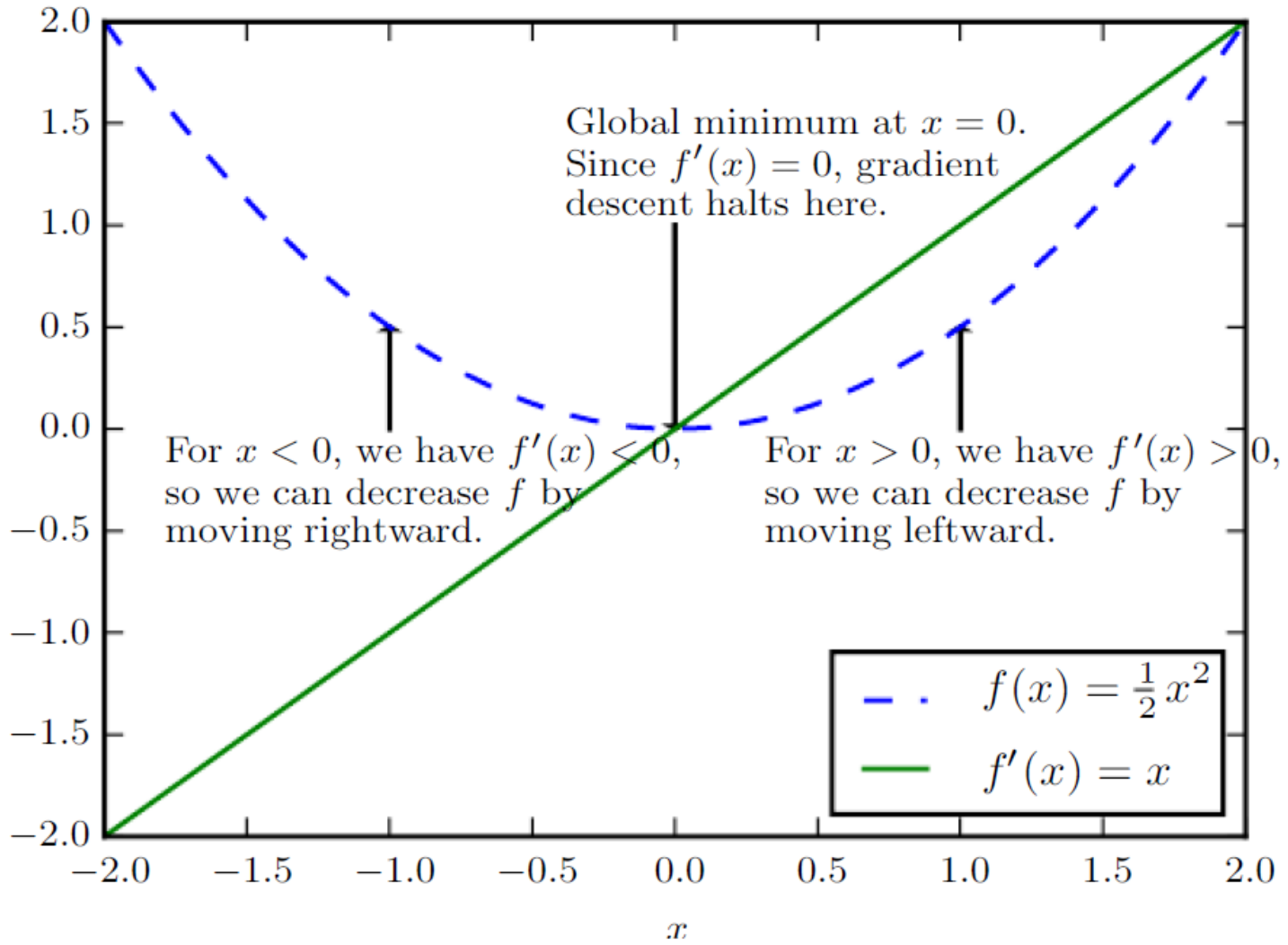


Figure 4.1: An illustration of how the gradient descent algorithm uses the derivatives of a function can be used to follow the function downhill to a minimum.

Gradient-Based Optimization

- When $f'(x) = 0$, the derivative provides no information about which direction to move. Points where $f'(x) = 0$ are known as **critical points** or **stationary points**.
- A **local minimum** is a point where $f(x)$ is lower than at all neighboring points, so it is no longer possible to decrease $f(x)$ by making infinitesimal steps.
- A **local maximum** is a point where $f(x)$ is higher than at all neighboring points, so it is not possible to increase $f(x)$ by making infinitesimal steps.
- Some critical points are neither maxima nor minima. These are known as **saddle points**.

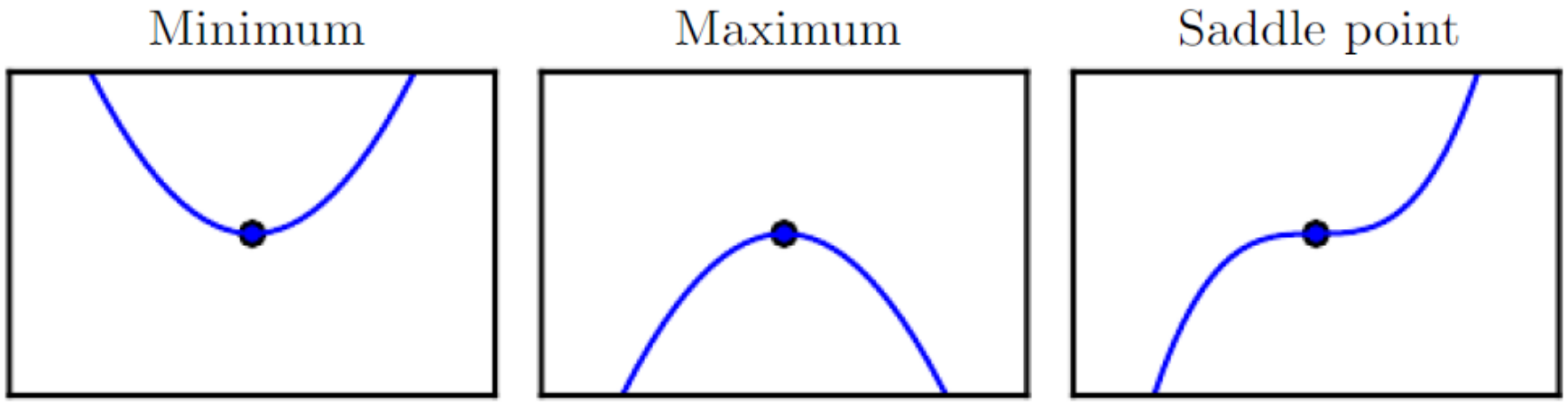


Figure 4.2: Examples of each of the three types of critical points in 1-D. A critical point is a point with zero slope. Such a point can either be a local minimum, which is lower than the neighboring points, a local maximum, which is higher than the neighboring points, or a saddle point, which has neighbors that are both higher and lower than the point itself.

Gradient-Based Optimization

- A point that obtains the absolute lowest value of $f(x)$ is a **global minimum**.
- It is possible for there to be only one global minimum or multiple global minima of the function.
- It is also possible for there to be local minima that are not globally optimal.
- In the context of deep learning, we optimize functions that may have many local minima that are not optimal, and many saddle points surrounded by very flat regions.
- All of this makes optimization very difficult, especially when the input to the function is multidimensional.
- We therefore usually settle for finding a value of f that is very low, but not necessarily minimal in any formal sense.

Gradient-Based Optimization

- We often minimize functions that have multiple inputs: $f : \mathrm{R}^n \rightarrow \mathrm{R}$.
- For the concept of “minimization” to make sense, there must still be only one (scalar) output.

Gradient-Based Optimization

- For functions with multiple inputs, we must make use of the concept of **partial derivatives**.
- The partial derivative $\partial/\partial x_i f(x)$ measures how f changes as only the variable x_i increases at point x.
- The **gradient** generalizes the notion of derivative to the case where the derivative is with respect to a vector: the gradient of f is the vector containing all of the partial derivatives, denoted $\nabla_x f(x)$.
- Element i of the gradient is the partial derivative of f with respect to x_i.

Gradient-Based Optimization

- In multiple dimensions, critical points are points where every element of the gradient is equal to zero.
- The **directional derivative** in direction u (a unit vector) is the slope of the function f in direction u.
- The directional derivative is the derivative of the function $f(x + \alpha u)$ with respect to α, evaluated at $\alpha = 0$.
- Using the chain rule, we can see that $\partial/\partial\alpha\, f(x + \alpha u)$ evaluates to $u^T \nabla_x f(x)$ when $\alpha = 0$.

$f: \mathbb{R}^n \rightarrow \mathbb{R}$

$f(x_1, x_2) = x_1^2 x_2$

∇_x, s.t $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ is $\begin{bmatrix} \partial/\partial x_1 \\ \partial/\partial x_2 \end{bmatrix}$

So, $\nabla_x \cdot f(x) = \nabla_x \cdot f(x_1, x_2)$

$= \left[\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}\right]$

$f(x + \alpha u) \Rightarrow f(x_1 + \alpha u_1, x_2 + \alpha u_2 \cdots)$

$\nabla_x = \begin{bmatrix} \partial/\partial x_1 \\ \partial/\partial x_2 \\ \vdots \\ i \end{bmatrix}$, $\nabla_\alpha f(x + \alpha u)$

$= \nabla_{x+\alpha u} f(x + \alpha u)$

$\nabla_\alpha (x + \alpha u)$

$= \begin{bmatrix} v_1, v_2 \cdots \end{bmatrix} \times \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ i \end{bmatrix}$

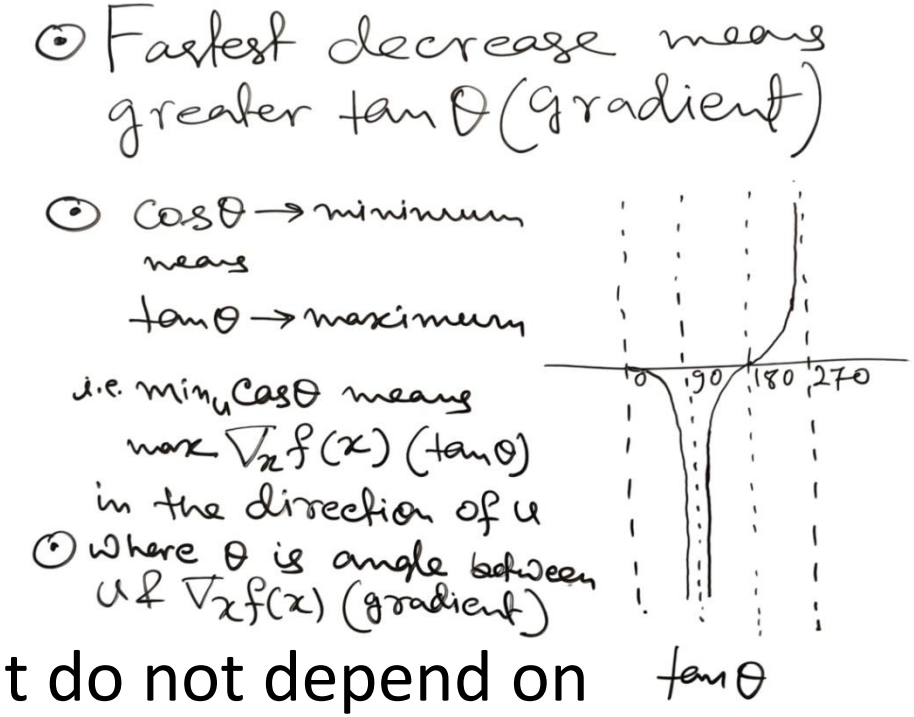
Gradient-Based Optimization

- To minimize *f*, we would like to find the direction in which *f* decreases the fastest.
- We can do this using the directional derivative:

$$\min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} \boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$$

$$= \min_{\boldsymbol{u}, \boldsymbol{u}^\top \boldsymbol{u}=1} ||\boldsymbol{u}||_2 ||\nabla_{\boldsymbol{x}} f(\boldsymbol{x})||_2 \cos\theta$$

- where ϑ is the angle between *u* and the gradient.
- Substituting in $||u||_2 = 1$ and ignoring factors that do not depend on *u*, this simplifies to $\min_u \cos \vartheta$.



Gradient-Based Optimization

- This is minimized when *u* points in the opposite direction as the gradient.
- In other words, the gradient points directly uphill, and the negative gradient points directly downhill.
- 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**.

Gradient-Based Optimization

- Steepest descent proposes a new point where ε is the **learning rate**, a positive scalar determining the size of the step.
- We can choose ε in several different ways.

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

- A popular approach is to set ε to a small constant.
- Sometimes, we can solve for the step size that makes the directional derivative vanish.
- Another approach is to evaluate $f(x - \varepsilon \nabla_x f(x))$ for several values of ε and choose the one that results in the smallest objective function value.
- This last strategy is called a **line search**.

Gradient-Based Optimization

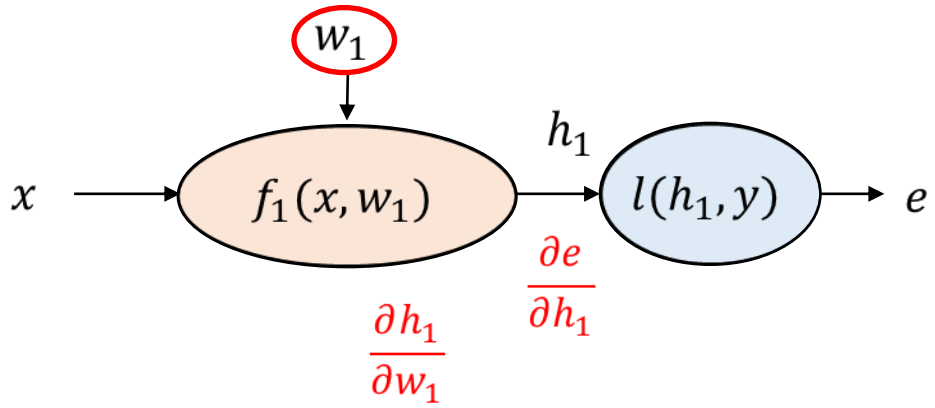
- Steepest descent converges when every element of the gradient is zero (or, in practice, very close to zero).
- We may be able to avoid running this iterative algorithm, and just jump directly to the critical point by solving the equation $\nabla_x f(x) = 0$ for x.
- Ascending an objective function of discrete parameters is called **hill climbing**

Gradient Vector Function: Optimization

- Sometimes we need to find all of the partial derivatives of a function whose input and output are both vectors.
- The matrix containing all such partial derivatives is known as a **Jacobian matrix**.
- Specifically, if we have a function $f : \mathrm{R}^m \rightarrow \mathrm{R}^n$, then the Jacobian matrix $J \in \mathrm{R}^{n \times m}$ of f is defined such that

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

Let's start with $k = 1$



$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_1}\frac{\partial h_1}{\partial w_1}$$

$$e = l(f_1(x, w_1), y)$$

$$\frac{\partial}{\partial w_1} l(f_1(x, w_1), y) = \frac{\partial e}{\partial h_1}\frac{\partial h_1}{\partial w_1}$$

$$\frac{\partial h_1}{\partial w_1} = x$$

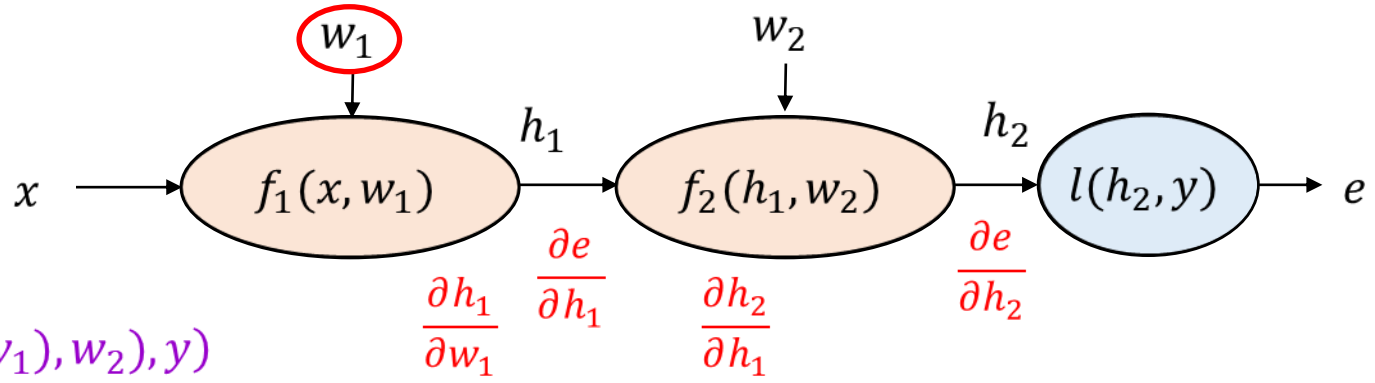
Example: $e = (y - w_1^T x)^2$

$h_1 = f_1(x, w_1) = w_1^T x$

$e = l(h_1, y) = (y - h_1)^2$

$$\frac{\partial e}{\partial h_1} = -2(y - h_1) = -2(y - w_1^T x)$$

$k = 2$



$e = l(f_2(f_1(x, w_1), w_2), y)$

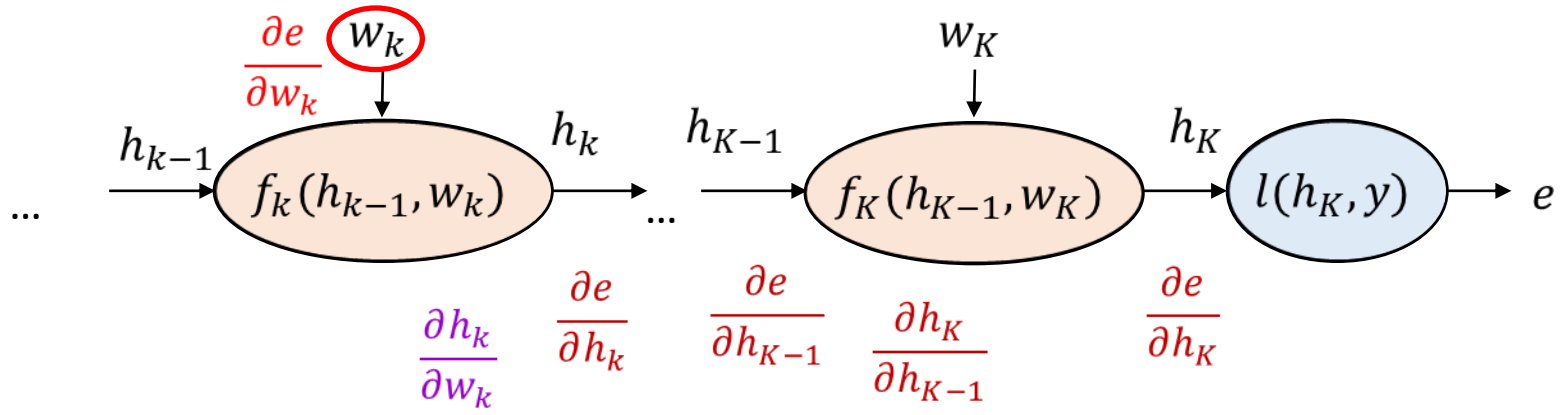
$$\frac{\partial e}{\partial w_2} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial w_2} \qquad \frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1}$$

Example: $e = -\log\left(\sigma(w_1^T x)\right)$ (assume $y = 1$)

$h_1 = f_1(x, w_1) = w_1^T x$
$h_2 = f_2(h_1) = \sigma(h_1)$
$e = l(h_2, 1) = -\log(h_2)$

$$\frac{\partial h_1}{\partial w_1} = x$$
$$\frac{\partial h_2}{\partial h_1} = \sigma'(h_1) = \sigma(h_1)(1 - \sigma(h_1))$$
$$\frac{\partial e}{\partial h_2} = -\frac{1}{h_2}$$

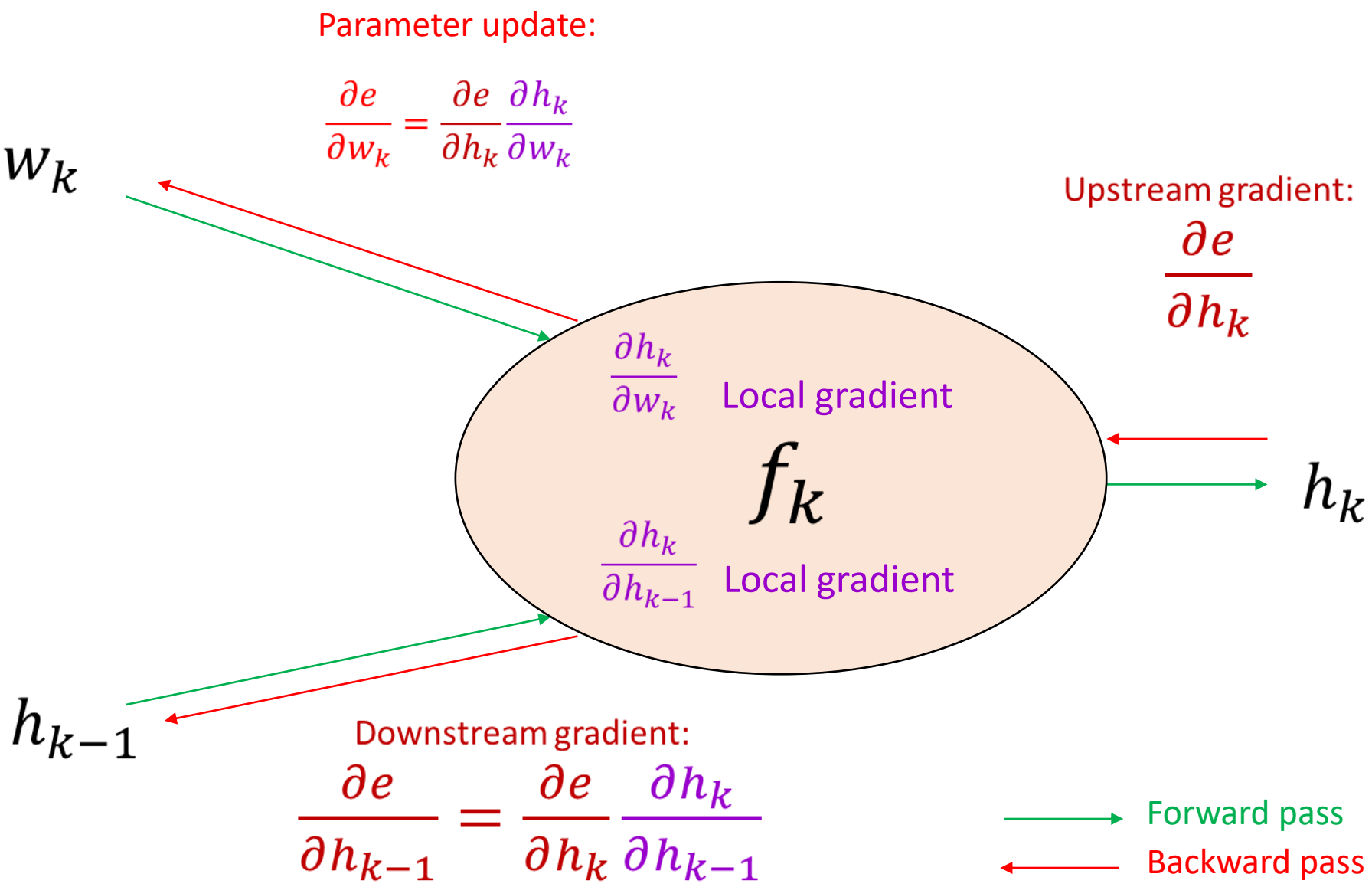
$$\frac{\partial e}{\partial w_1} = \frac{\partial e}{\partial h_2}\frac{\partial h_2}{\partial h_1}\frac{\partial h_1}{\partial w_1} = -\frac{1}{\sigma(w_1^T x)}\sigma(w_1^T x)\left(1 - \sigma(w_1^T x)\right)x =$$
$$-\sigma(-w_1^T x)x$$



- General case:

- $$\frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_K} \boxed{\frac{\partial h_K}{\partial h_{K-1}} \cdots \frac{\partial h_{k+1}}{\partial h_k} \frac{\partial h_k}{\partial w_k}}$$

Upstream gradient $\frac{\partial e}{\partial h_k}$ Local gradient


Parameter update:
$\frac{\partial e}{\partial w_k} = \frac{\partial e}{\partial h_k}\frac{\partial h_k}{\partial w_k}$
w_k
Upstream gradient:
$\frac{\partial e}{\partial h_k}$
$\frac{\partial h_k}{\partial w_k}$ Local gradient
f_k
$\frac{\partial h_k}{\partial h_{k-1}}$ Local gradient
h_k
h_{k-1}
Downstream gradient:
$\frac{\partial e}{\partial h_{k-1}} = \frac{\partial e}{\partial h_k}\frac{\partial h_k}{\partial h_{k-1}}$
Forward pass
Backward pass

$$\frac{\partial \text{error}}{\partial w_k^{(i)}} = \frac{\partial \text{error}}{\partial z_k^{(1)}} \cdot \frac{\partial z_k^{(1)}}{\partial w_k^{(i)}} + \frac{\partial \text{error}}{\partial z_k^{(2)}} \cdot \frac{\partial z_k^{(2)}}{\partial w_k^{(i)}} + \frac{\partial \text{error}}{\partial z_k^{(3)}} \cdot \frac{\partial z_k^{(3)}}{\partial w_k^{(i)}} + \dots$$

$$\frac{\partial \text{error}}{\partial w_k^{(i)}} = \sum_j \frac{\partial \text{error}}{\partial z_k^{(j)}} \cdot \frac{\partial z_k^{(j)}}{\partial w_k^{(i)}}$$

$$\nabla_{\mathbf{w}_k} \text{error} = \left(\frac{\partial \mathbf{z}_k}{\partial \mathbf{w}_k} \right)^{\mathsf{T}} \nabla_{\mathbf{z}_k} \text{error}$$

$$f : \mathbb{R}^u \to \mathbb{R}$$

$$\mathbf{J} = \frac{df(x)}{dx} = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \cdots \frac{\partial f(x)}{\partial x_u} \end{bmatrix}$$

$$\mathbf{f} : \mathbb{R}^u \to \mathbb{R}^v$$

$$\mathbf{J} = \frac{d\mathbf{f}(\mathbf{x})}{d\mathbf{x}} = \begin{bmatrix} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_1} \cdots \frac{\partial \mathbf{f}(\mathbf{x})}{\partial x_u} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_u} \\ \vdots & & \vdots \\ \frac{\partial f_v(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_v(\mathbf{x})}{\partial x_u} \end{bmatrix}$$

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial z_k^{(1)}}{\partial w_k^{(1)}} & \cdots & \dfrac{\partial z_k^{(1)}}{\partial w_k^{(u)}} \\ \vdots & & \vdots \\ \dfrac{\partial z_k^{(v)}}{\partial w_k^{(1)}} & \cdots & \dfrac{\partial z_k^{(v)}}{\partial w_k^{(u)}} \end{bmatrix}$$

Gradient Vector Function: Optimization

- We are also sometimes interested in a derivative of a derivative. This is known as a **second derivative.**
- For example, for a function $f : R^n \rightarrow R$, the derivative with respect to x_i of the derivative of f with respect to x_j is denoted as $\frac{\partial^2}{\partial x_i \partial x_j} f$.

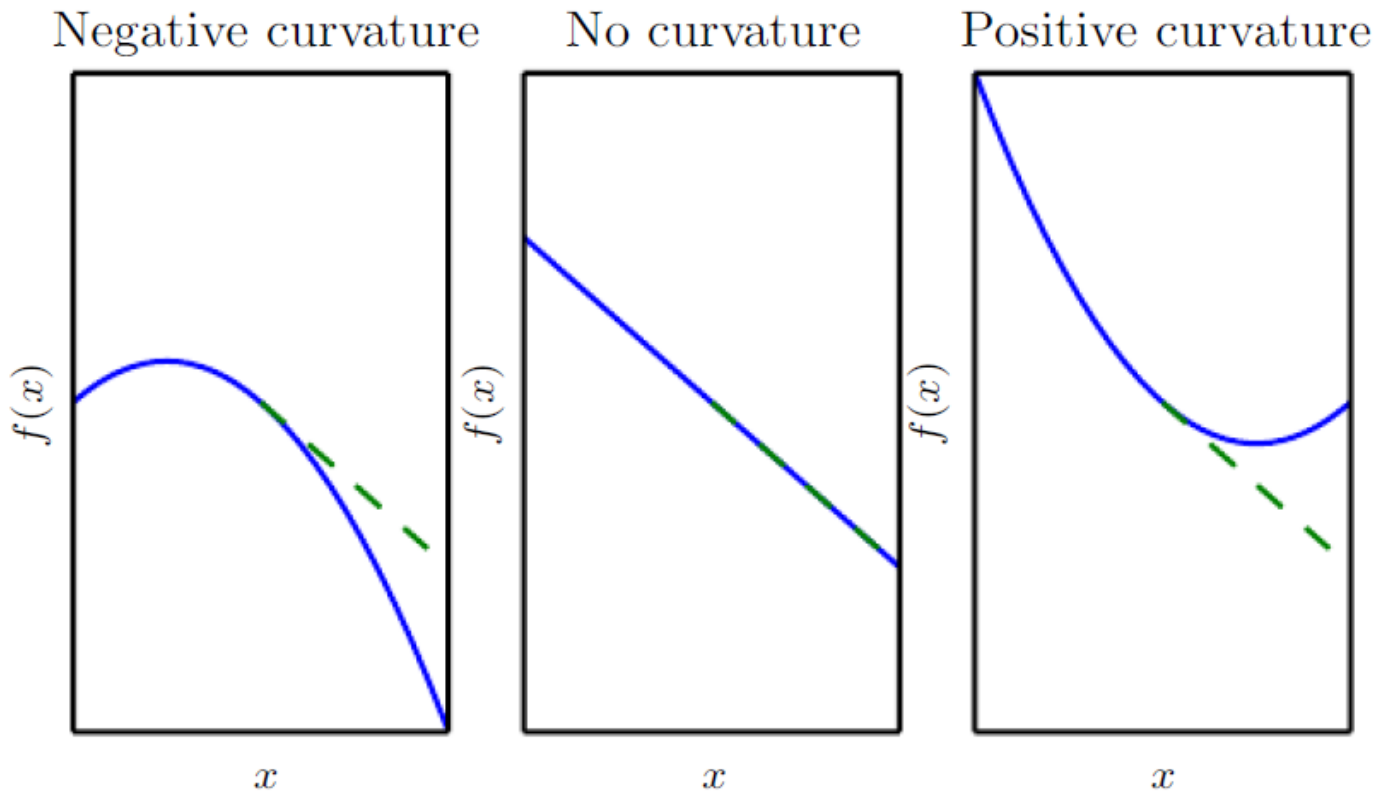
Gradient Vector Function: Optimization

- The second derivative tells us how the first derivative will change as we vary the input.
- This is 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.
- We can think of the second derivative as measuring **curvature.**
- Suppose we have a quadratic function (many functions that arise in practice are not quadratic but can be approximated well as quadratic, at least locally).
- If such a function has a second derivative of zero, then there is no curvature.
- It is a perfectly flat line, and its value can be predicted using only the gradient.
- If the gradient is 1, then we can make a step of size ϵ along the negative gradient, and the cost function will decrease by ϵ.

Gradient Vector Function: Optimization

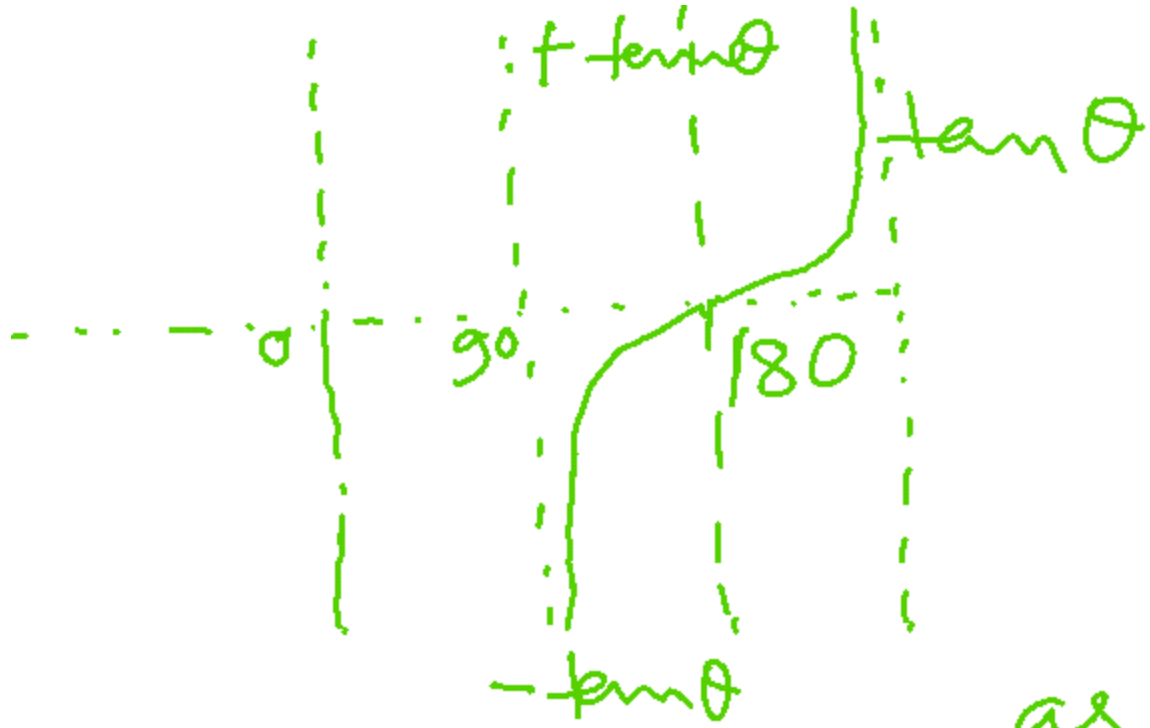
- If the second derivative is negative, the function curves downward, so the cost function will actually decrease by more than *ε*.
- Finally, if the second derivative is positive, the function curves upward, so the cost function can decrease by less than *ε*.

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



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

Figure 4.4: The second derivative determines the curvature of a function. Here we show quadratic functions with various curvature. The dashed line indicates the value of the cost function we would expect based on the gradient information alone as we make a gradient step downhill. In the case of negative curvature, the cost function actually decreases faster than the gradient predicts. In the case of no curvature, the gradient predicts the decrease correctly. In the case of positive curvature, the function decreases slower than expected and eventually begins to increase, so steps that are too large can actually increase the function inadvertently.



as θ increases beyond 90° value of tanθ increases as magnitude decreases.

Fig-1

⊙ As θ increases tanθ becomes less -ve. i.e magnitude decreases

⊙ If θ decreases tanθ becomes more -ve. i.e magnitude increases.

$$x_1 = x_0 + \epsilon \left(\frac{\partial f(x)}{\partial x}\right)_{x_0}$$

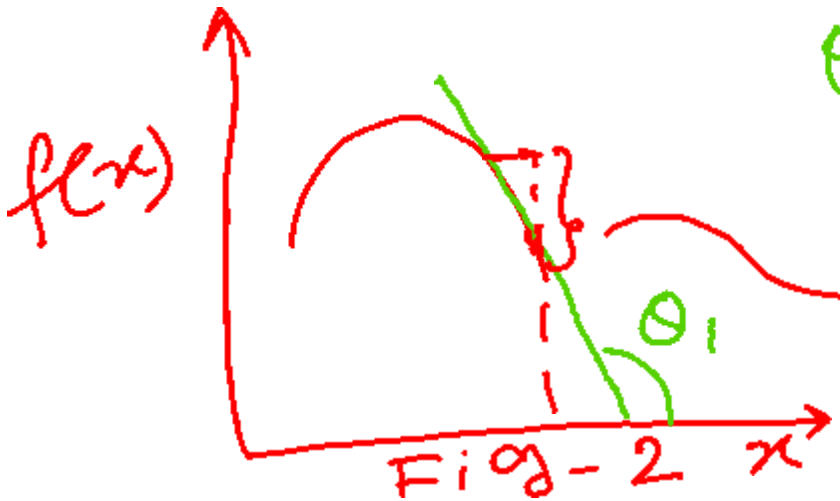


Fig-2

$\theta > \theta_1$, So $\tan\theta_1$ has larger magnitude.

change in f(x) is more for small change in x

In fig-1 shown in previous slide the θ is more so the magnitude of $\left(\frac{\partial f(x)}{\partial x}\right)_{x_0}$ is small so the step size is small.

In fig-2, θ suddenly changes to small θ value i.e. $f(x)$ falls more for small step from x_0.

Gradient Vector Function: Optimization

- When our function has multiple input dimensions, there are many second derivatives.
- These derivatives can be collected together into a matrix called the **Hessian matrix.**
- The Hessian matrix H(f)(x) is defined such that

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

Equivalently, the Hessian is the Jacobian of the gradient

Gradient Vector Function: Optimization

Anywhere that the second partial derivatives are continuous, the differential operators are commutative, i.e. their order can be swapped:

$$\frac{\partial^2}{\partial x_i \partial x_j} f(\boldsymbol{x}) = \frac{\partial^2}{\partial x_j \partial x_i} f(\boldsymbol{x}). \tag{4.7}$$

This implies that $H_{i,j} = H_{j,i}$, so the Hessian matrix is symmetric at such points.

Gradient Vector Function: Optimization

- Because the Hessian matrix is real and symmetric, we can decompose it into a set of real Eigenvalues and an orthogonal basis of Eigenvectors.
- The second derivative in a specific direction represented by a unit vector *d* is given by $d^T Hd$.
- When *d* is an eigenvector of *H,* the second derivative in that direction is given by the corresponding Eigenvalue.
- For other directions of *d*, the directional second derivative is a weighted average of all of the Eigenvalues, with weights between 0 and 1, and eigenvectors that have smaller angle with *d* receiving more weight.
- The maximum Eigenvalue determines the maximum second derivative and the minimum Eigenvalue determines the minimum second derivative.

An **eigenvector** of a square matrix $\boldsymbol{A}$ is a non-zero vector $\boldsymbol{v}$ such that multiplication by $\boldsymbol{A}$ alters only the scale of $\boldsymbol{v}$: $\boldsymbol{A}\boldsymbol{v} = \lambda\boldsymbol{v}$.

$$\boldsymbol{v}^\top \boldsymbol{A} = \lambda \boldsymbol{v}^\top$$

Gradient Vector Function: Optimization

The (directional) second derivative tells us how well we can expect a gradient descent step to perform. We can make a second-order Taylor series approximation to the function $f(\boldsymbol{x})$ around the current point $\boldsymbol{x}^{(0)}$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{g} + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(\boldsymbol{x} - \boldsymbol{x}^{(0)}). \tag{4.8}$$

where $\boldsymbol{g}$ is the gradient and $\boldsymbol{H}$ is the Hessian at $\boldsymbol{x}^{(0)}$. If we use a learning rate of ϵ, then the new point $\boldsymbol{x}$ will be given by $\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}$. Substituting this into our approximation, we obtain

$$f(\boldsymbol{x}^{(0)} - \epsilon \boldsymbol{g}) \approx f(\boldsymbol{x}^{(0)}) - \epsilon \boldsymbol{g}^\top \boldsymbol{g} + \frac{1}{2}\epsilon^2 \boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}. \tag{4.9}$$

Gradient Vector Function: Optimization

- There are three terms here: the original value of the function, the expected improvement due to the slope of the function, and the correction we must apply to account for the curvature of the function.
- When this last term is too large, the gradient descent step can actually move uphill.

Gradient Vector Function: Optimization

- When $g^T Hg$ *is zero or negative,* the Taylor series approximation predicts that increasing ε forever will decrease f forever.
- In practice, the Taylor series is unlikely to remain accurate for large ε, *so* one must resort to more heuristic choices of ε in this case.
- *When* $g^T Hg$ *is positive,* solving for the optimal step size that decreases the Taylor series approximation of the function the most yields

$$\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H} \boldsymbol{g}}$$

Gradient Vector Function: Optimization

- In the worst case, when *g* aligns with the eigenvector of *H* corresponding to the maximal Eigenvalue λ_{max}, then this optimal step size is given by $1/\lambda_{max}$
- The Eigenvalues of the Hessian thus determine the scale of the learning rate.

Gradient Vector Function: Optimization

- The second derivative can be used to determine whether a critical point is a local maximum, a local minimum, or saddle point.
- Recall that on a critical point, $f'(x) = 0$. When the second derivative $f''(x)>0$, the first derivative $f'(x)$ increases as we move to the right and decreases as we move to the left.

Gradient Vector Function: Optimization

- This means $f'(x - \varepsilon) < 0$ *and* $f'(x + \varepsilon) > 0$ for small enough ε.
- In other words, as we move right, the slope begins to point uphill to the right, and as we move left, the slope begins to point uphill to the left.
- Thus, when $f'(x) = 0$ *and* $f''(x) > 0$, we can conclude that x is a local minimum.
- Similarly, when $f'(x) = 0$ and $f''(x) < 0$, we can conclude that x is a local maximum. This is known as the **second derivative test.**
- Unfortunately, when $f''(x) = 0$, *the test is inconclusive.* In this case x may be a saddle point, or a part of a flat region.

Gradient Vector Function: Optimization

- In multiple dimensions, we need to examine all of the second derivatives of the function.
- Using the Eigen decomposition of the Hessian matrix, we can generalize the second derivative test to multiple dimensions.
- At a critical point, where $\nabla_x f(x) = 0$, we can examine the Eigenvalues of the Hessian to determine whether the critical point is a local maximum, local minimum, or saddle point.
- When the Hessian is positive definite (all its Eigenvalues are positive), the point is a local minimum.
- This can be seen by observing that the directional second derivative in any direction must be positive, and making reference to the univariate second derivative test.

Gradient Vector Function: Optimization

- Likewise, when the Hessian is negative definite (all its Eigenvalues are negative), the point is a local maximum.
- In multiple dimensions, it is actually possible to find positive evidence of saddle points in some cases.
- When at least one Eigenvalue is positive and at least one Eigenvalue is negative, we know that *x* is a local maximum on one cross section of *f* but a local minimum on another cross section.

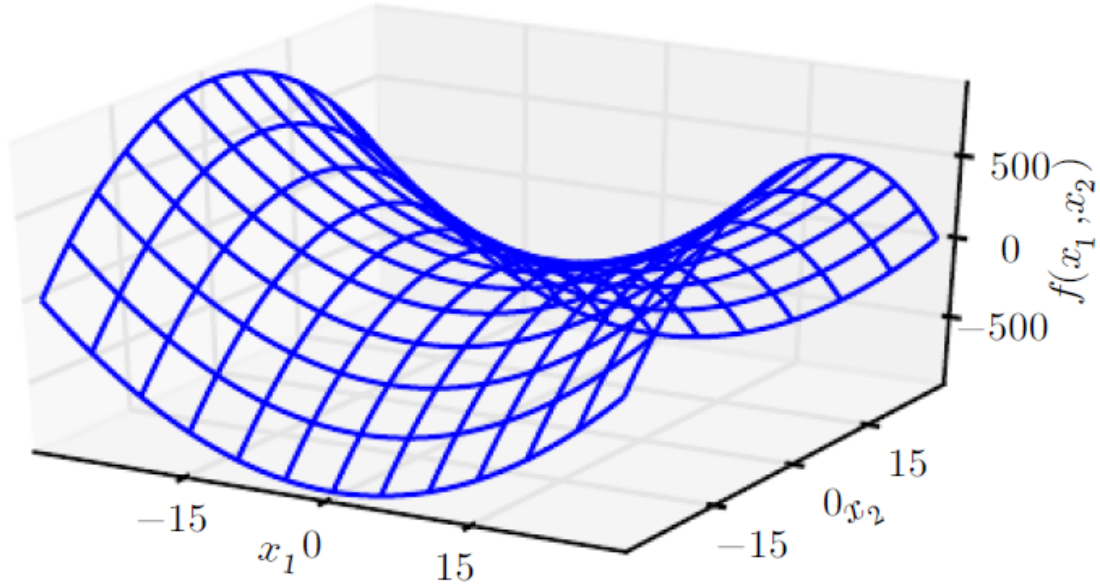


Figure 4.5: A saddle point containing both positive and negative curvature. The function in this example is $f(\boldsymbol{x}) = x_1^2 - x_2^2$. Along the axis corresponding to x_1, the function curves upward. This axis is an eigenvector of the Hessian and has a positive eigenvalue. Along the axis corresponding to x_2, the function curves downward. This direction is an eigenvector of the Hessian with negative eigenvalue. The name "saddle point" derives from the saddle-like shape of this function. This is the quintessential example of a function with a saddle point. In more than one dimension, it is not necessary to have an eigenvalue of 0 in order to get a saddle point: it is only necessary to have both positive and negative eigenvalues. We can think of a saddle point with both signs of eigenvalues as being a local maximum within one cross section and a local minimum within another cross section.

Gradient Vector Function: Optimization

- Finally, the multidimensional second derivative test can be inconclusive, just like the univariate version.
- The test is inconclusive whenever all of the non-zero Eigenvalues have the same sign, but at least one Eigenvalue is zero.
- This is because the univariate second derivative test is inconclusive in the cross section corresponding to the zero Eigenvalue.

Gradient Vector Function: Optimization

- In multiple dimensions, there is a different second derivative for each direction at a single point.
- The condition number of the Hessian at this point measures how much the second derivatives differ from each other.
- When the Hessian has a poor condition number, gradient descent performs poorly.
- This is because in one direction, the derivative increases rapidly, while in another direction, it increases slowly.
- Gradient descent is unaware of this change in the derivative so it does not know that it needs to explore preferentially in the direction where the derivative remains negative for longer.

Gradient Vector Function: Optimization

- It also makes it difficult to choose a good step size.
- The step size must be small enough to avoid overshooting the minimum and going uphill in directions with strong positive curvature.
- This usually means that the step size is too small to make significant progress in other directions with less curvature.
- This issue can be resolved by using information from the Hessian matrix to guide the search

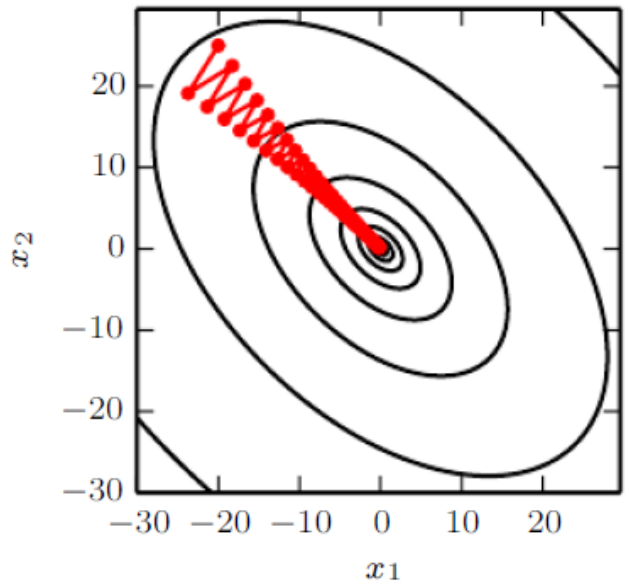


Figure 4.6: Gradient descent fails to exploit the curvature information contained in the Hessian matrix. Here we use gradient descent to minimize a quadratic function $f(\boldsymbol{x})$ whose Hessian matrix has condition number 5. This means that the direction of most curvature has five times more curvature than the direction of least curvature. In this case, the most curvature is in the direction $[1, 1]^\top$ and the least curvature is in the direction $[1, -1]^\top$. The red lines indicate the path followed by gradient descent. This very elongated quadratic function resembles a long canyon. Gradient descent wastes time repeatedly descending canyon walls, because they are the steepest feature. Because the step size is somewhat too large, it has a tendency to overshoot the bottom of the function and thus needs to descend the opposite canyon wall on the next iteration. The large positive eigenvalue of the Hessian corresponding to the eigenvector pointed in this direction indicates that this directional derivative is rapidly increasing, so an optimization algorithm based on the Hessian could predict that the steepest direction is not actually a promising search direction in this context.

Gradient Vector Function: Optimization

- The simplest method for doing so is known as **Newton's method.**
- Newton's method is based on using a second-order Taylor series expansion to approximate *f(x) near some point* $x^{(0)}$*:*

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)}). \quad (4.11)$$

If we then solve for the critical point of this function, we obtain:

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}). \quad (4.12)$$

The 2nd Taylor approximation of $f(x)$ at a point $x = a$

$$P(x) = f(a) + f'(a)(x-a)^1 + \frac{1}{2} f''(a)(x-a)^2$$

$$f(x) \approx f(x_0) + (x - x_0)^T \underbrace{\nabla_x f(x_0)}_{g} + \tfrac{1}{2}(x - x_0)^T \underbrace{H(f)(x_0)}_{H}(x - x_0)$$

$$f(x) \approx f(x_0) + (x - x_0)^T g + \tfrac{1}{2}(x - x_0)^T H (x - x_0)$$

$$\tfrac{1}{2}\{x^T H x - x_0^T H x - \underbrace{x^T H x_0}_{x_0^T H x} + x_0^T H x_0\}$$

$$\tfrac{1}{2}\{x^T H x - 2x_0^T H x + x_0^T H x_0\}$$

$$f(x) \approx f(x_0) + x^T g - x_0^T g + \tfrac{1}{2} x^T H x - x_0^T H x + \tfrac{1}{2} x_0^T H x_0$$

$$\approx \tfrac{1}{2} x^T H x + (x^T g - x_0^T H x) + \underbrace{f(x_0) - x_0^T g + \tfrac{1}{2} x_0^T H x_0}_{c}$$

$$\approx \tfrac{1}{2} x^T H x + x^T (g - H x_0) + c$$

$$\approx \tfrac{1}{2} x^T H x + x^T b + c$$

$$\nabla f(x) = Hx + b \quad \therefore \quad x = -H^{-1} b = -H^{-1}(g - H x_0)$$

Calculate the differential of the function $f : \mathbb{R}^n \to \mathbb{R}$ given by

$$f(x) = x^T A x$$

with A symmetric.

Let $\mathbf{x}^{n\times 1} = (x_1, \ldots, x_n)'$ be a vector, the derivative of $\mathbf{y} = f(\mathbf{x})$ with respect to the vector $\mathbf{x}$ is defined by

$$\frac{\partial f}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}$$

$$
\begin{aligned}
\mathbf{y} &= f(\mathbf{x}) \\
&= \mathbf{x}' A \mathbf{x} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij} x_i x_j \\
&= \sum_{i=1}^{n} a_{i1} x_i x_1 + \sum_{j=1}^{n} a_{1j} x_1 x_j + \sum_{i=2}^{n} \sum_{j=2}^{n} a_{ij} x_i x_j \\
\frac{\partial f}{\partial x_1} &= \sum_{i=1}^{n} a_{i1} x_i + \sum_{j=1}^{n} a_{1j} x_j \\
&= \sum_{i=1}^{n} a_{1i} x_i + \sum_{i=1}^{n} a_{1i} x_i \, [\text{since } a_{i1} = a_{1j}] \\
&= 2 \sum_{i=1}^{n} a_{1i} x_i \\
\frac{\partial f}{\partial \mathbf{x}} &= \begin{pmatrix} 2 \sum_{i=1}^{n} a_{1i} x_i \\ \vdots \\ 2 \sum_{i=1}^{n} a_{ni} x_i \end{pmatrix} = 2 \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\
&= 2A\mathbf{x}
\end{aligned}
$$

Gradient Vector Function: Optimization

- Optimization algorithms that use only the gradient, such as gradient descent, are called **first-order optimization algorithms.**
- Optimization algorithms that also use the Hessian matrix, such as Newton's method, are called **second-order optimization algorithms**

Gradient Vector Function: Optimization

- In the context of deep learning, we sometimes gain some guarantees by restricting ourselves to functions that are either **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**

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

Constrained Optimization

- Sometimes we wish not only to maximize or minimize a function *f(x)* over all possible values of *x.*
- Instead we may wish to find the maximal or minimal value of f(x) for values of x in some set S. This is known as **constrained optimization.**
- Points x that lie within the set S are called feasible points in constrained optimization terminology.
- We often wish to find a solution that is small in some sense. A common approach in such situations is to impose a norm constraint, such as $||x|| \leq 1$.

Constrained Optimization

- One simple approach to constrained optimization is simply to modify gradient descent taking the constraint into account.
- If we use a small constant step size ϵ, we can make gradient descent steps, then project the result back into S.
- If we use a line search, we can search only over step sizes ϵ that yield new x points that are feasible, or we can project each point on the line back into the constraint region.
- When possible, this method can be made more efficient by projecting the gradient into the tangent space of the feasible region before taking the step or beginning the line search

Constrained Optimization

- A more sophisticated approach is to design a different, unconstrained optimization problem whose solution can be converted into a solution to the original, constrained optimization problem.
- For example, if we want to minimize $f(x)$ for $x \in R^n$ with x constrained to have exactly unit L2 norm, we can instead minimize $g(\theta) = f([\cos \theta, \sin \theta]^T)$ with respect to θ, then return $[\cos\theta, \sin\theta]$ as the solution to the original problem.
- This approach requires creativity; the transformation between optimization problems must be designed specifically for each case we encounter.

Constrained Optimization

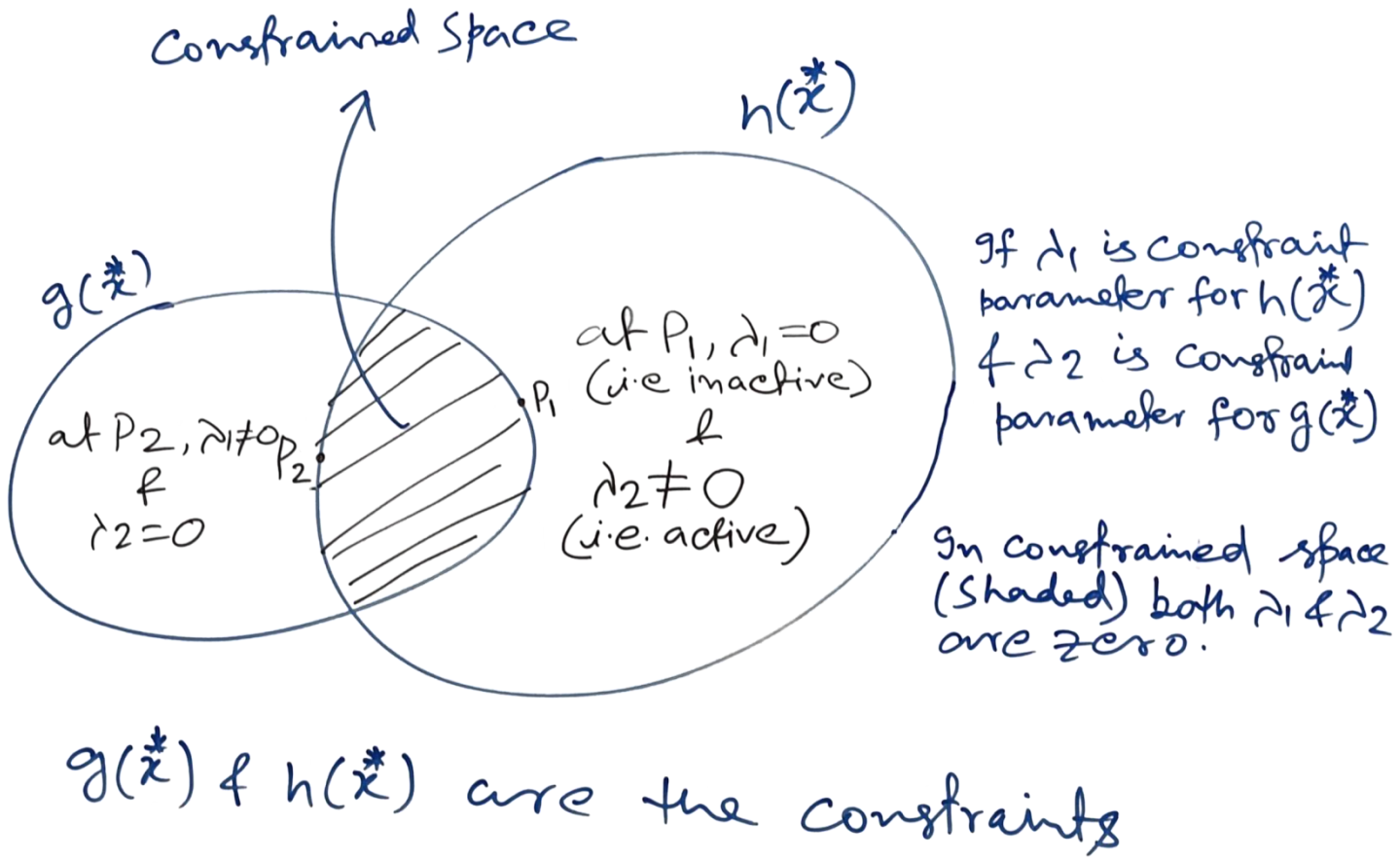
- **Generalized Lagrange function**
  - To define the Lagrangian, we first need to describe S in terms of equations and inequalities.
  - We want a description of S in terms of m functions $g^{(i)}$ and n functions $h^{(j)}$ so that $S = \{x \mid \forall_i, g^{(i)}(x) = 0 \text{ and } \forall_j, h^{(j)}(x) \leq 0\}$.
  - The equations involving $g^{(i)}$ are called the **equality constraints** and the inequalities involving $h^{(j)}$ are called **inequality constraints.**

Constrained Optimization

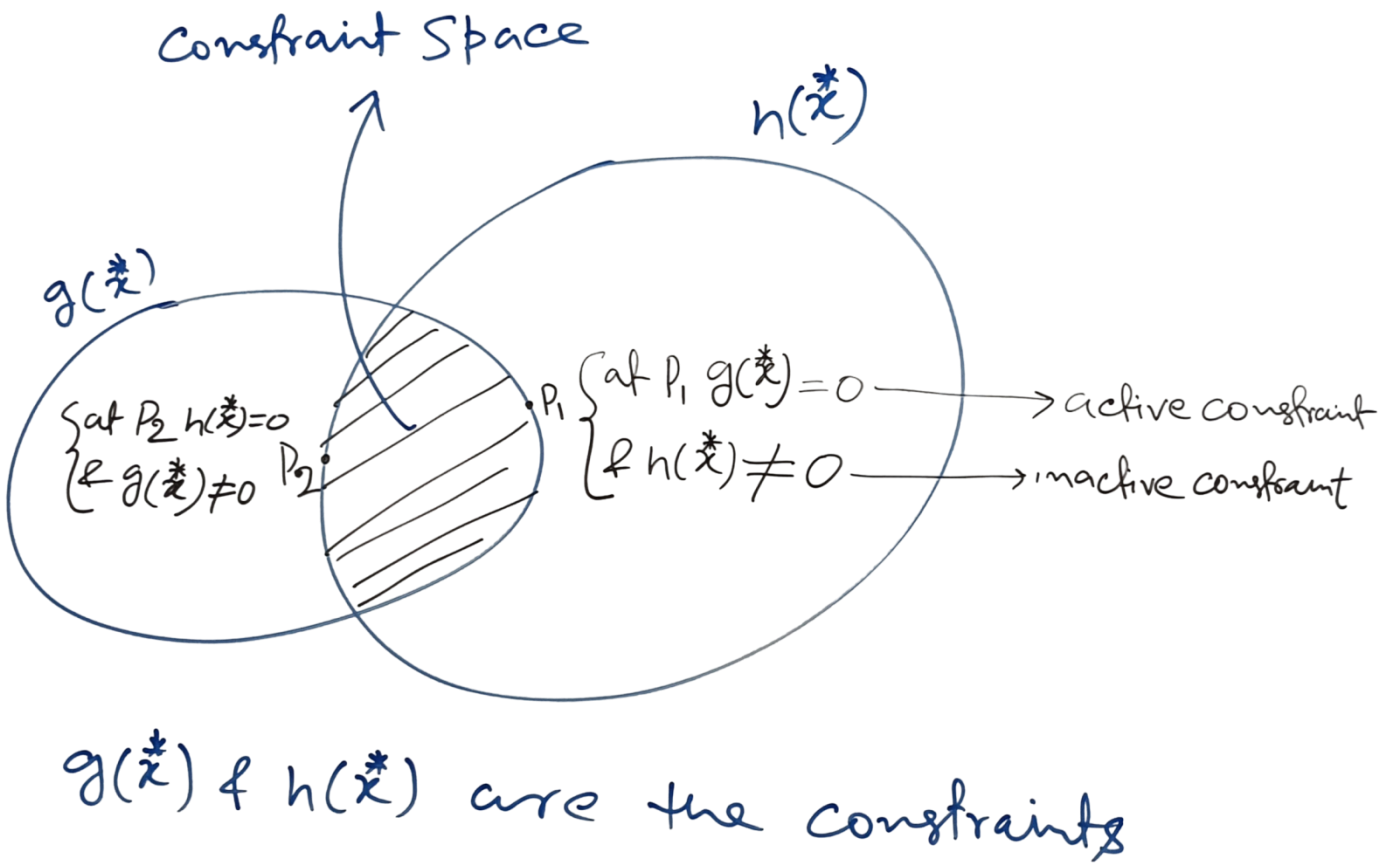
- Variables λ_i *and* α_j for each constraint are called the KKT multipliers. The generalized Lagrangian is then defined as

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

Constrained Optimization

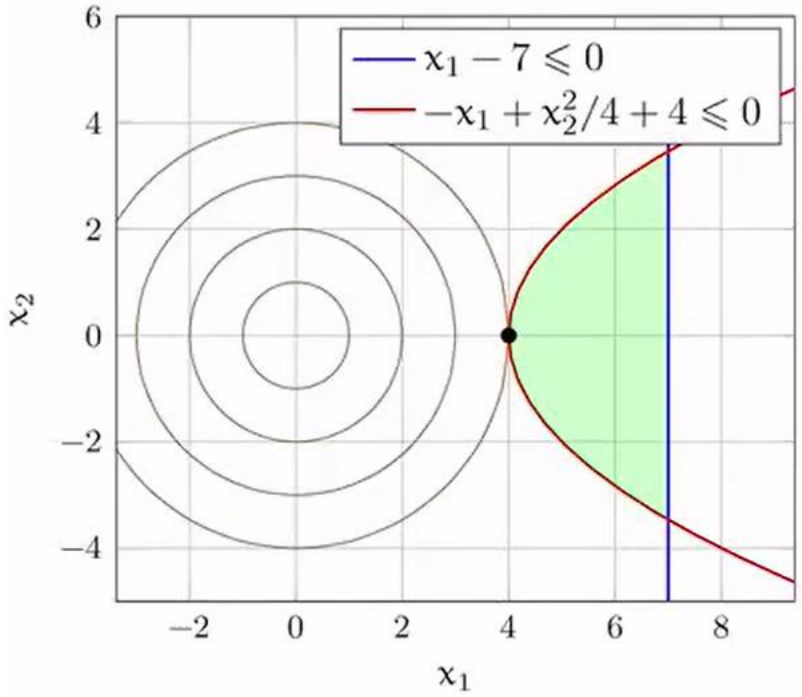


Constrained Optimization



Constrained Optimization

Example 4.14. Minimise the objective function $f(\boldsymbol{x}) = x_1^2 + x_2^2$, which has $\nabla f(\boldsymbol{x}) = (2x_1, 2x_2)$, subject to constraints $g_1(\boldsymbol{x}) = x_1 - 7 \leq 0$ and $g_2(\boldsymbol{x}) = -x_1 + x_2^2/4 + 4 \leq 0$.



Here $g_2(x)$ is active, hence λ_2 corresponding to $g_2(x)$ will be non zero.

$$\mathcal{L}(x) = f(x) + \lambda_1 g_1(x) + \lambda_2 g_2(x)$$

here, $\lambda_1 = 0$ & $\lambda_2 = ?$

$$\nabla\mathcal{L}(x) = \nabla f(x) + \lambda_2 \nabla g_2(x) = 0$$

$$\therefore \nabla\mathcal{L}(x) = \begin{bmatrix} 2x_1 \\ 2x_2 \end{bmatrix} + \lambda_2 \begin{bmatrix} -1 \\ \frac{x_2}{2} \end{bmatrix} = 0$$

$$\therefore 2x_1 - \lambda_2 = 0 \quad \& \quad 2x_2 + \frac{\lambda_2 x_2}{2} = 0$$

The point at which $g_2(x)$ is active is $(4, 0)$.

$\therefore x_1 = 4$, hence $\lambda_2 = 2x_1 = 8$

Constrained Optimization

- Observe that, so long as at least one feasible point exists and f(x) is not permitted to have value ∞, then

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

  has the same optimal objective function value and set of optimal points x as

$$\min_{\boldsymbol{x} \in \mathbb{S}} f(\boldsymbol{x})$$

Constrained Optimization

This follows because any time the constraints are satisfied,

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

while any time a constraint is violated,

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

- These properties guarantee that no infeasible point can be optimal, and that the optimum within the feasible points is unchanged.

Constrained Optimization

To perform constrained maximization, we can construct the generalized Lagrange function of $-f(\boldsymbol{x})$, which leads to this optimization problem:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} -f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.19}$$

Constrained Optimization

To perform constrained maximization, we can construct the generalized Lagrange function of $-f(\boldsymbol{x})$, which leads to this optimization problem:

$$\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} -f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) + \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.19}$$

We may also convert this to a problem with maximization in the outer loop:

$$\max_{\boldsymbol{x}} \min_{\boldsymbol{\lambda}} \min_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} f(\boldsymbol{x}) + \sum_i \lambda_i g^{(i)}(\boldsymbol{x}) - \sum_j \alpha_j h^{(j)}(\boldsymbol{x}). \tag{4.20}$$

The sign of the term for the equality constraints does not matter; we may define it with addition or subtraction as we wish, because the optimization is free to choose any sign for each λ_i.

Constrained Optimization

- The inequality constraints are particularly interesting.
- We say that a constraint $h^{(i)}(x)$ is active if $h^{(i)}(x^*) = 0$. If a constraint is not active, then the solution to the problem found using that constraint would remain at least a local solution if that constraint were removed.

Constrained Optimization

- It is possible that an inactive constraint excludes other solutions.
- For example, a convex problem with an entire region of globally optimal points (a wide, flat, region of equal cost) could have a subset of this region eliminated by constraints, or a non-convex problem could have better local stationary points excluded by a constraint that is inactive at convergence.
- The point found at convergence remains a stationary point whether or not the inactive constraints are included.
- Because an inactive $h^{(i)}$ has negative value, then the solution to $\min_x \max_\lambda \max_{\alpha,\alpha\geq 0} L(x,\lambda,\alpha)$ will have $\alpha_i = 0$

Constrained Optimization

- We can thus observe that at the solution, $\alpha \odot h(x) = 0$.
- In other words, for all i, we know that at least one of the constraints $\alpha_i \geq 0$ and $h^{(i)}(x) \leq 0$ must be active at the solution.
- To gain some intuition for this idea, we can say that either the solution is on the boundary imposed by the inequality and we must use its KKT multiplier to influence the solution to x, or the inequality has no influence on the solution and we represent this by zeroing out its KKT multiplier.

Constrained Optimization

Suppose we want to find the value of $\boldsymbol{x}$ that minimizes

$$f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2. \tag{4.21}$$

There are specialized linear algebra algorithms that can solve this problem efficiently. However, we can also explore how to solve it using gradient-based optimization as a simple example of how these techniques work.

First, we need to obtain the gradient:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{A}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top\boldsymbol{b}. \tag{4.22}$$

$$\frac{1}{2}\|Ax-b\|^2 = \frac{1}{2}(Ax-b)^T(Ax-b) = \frac{1}{2}(x^TA^TAx - 2x^TA^Tb + b^Tb).$$

$$\frac{\partial x^TA^TAx}{\partial x} = 2A^TAx.$$

$$\frac{\partial x^TA^Tb}{\partial x} = A^Tb.$$

$$\frac{\partial}{\partial x}\frac{1}{2}\|Ax-b\|^2 = \frac{1}{2}(2A^TAx - 2A^Tb) = A^T(Ax-b)$$

Constrained Optimization

$$\begin{bmatrix} x_1 & x_2 \end{bmatrix} \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} x_1 a_1 + x_2 a_3 & x_1 a_2 + x_2 a_4 \end{bmatrix} \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} x_1 a_1 a_1 + x_2 a_3 a_1 + x_1 a_2 a_2 + x_2 a_4 a_2 & x_1 a_1 a_3 + x_2 a_3 a_3 + x_1 a_2 a_4 + x_2 a_4 a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} x_1^2 a_1 a_1 + x_2 x_1 a_3 a_1 + x_1^2 a_2 a_2 + x_2 x_1 a_4 a_2 + x_1 x_2 a_1 a_3 + x_2^2 a_3 a_3 + x_1 x_2 a_2 a_4 + x_2^2 a_4 a_4 \end{bmatrix}$$

where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $A = \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix}$

Constrained Optimization

Let $F = x_1^2 a_1 a_1 + x_2 x_1 a_3 a_1 + x_1^2 a_2 a_2 + x_2 x_1 a_4 a_2 + x_1 x_2 a_1 a_3 + x_2^2 a_3 a_3 + x_1 x_2 a_2 a_4 + x_2^2 a_4 a_4$

$\therefore \; x^T A^T A x = F$

$$\Rightarrow \nabla_x (x^T A^T A x) = \begin{bmatrix} \frac{\partial F}{\partial x_1} \\ \frac{\partial F}{\partial x_2} \end{bmatrix}$$

where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $A = \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix}$

Constrained Optimization

$$\nabla_x (x^T A^T A x) = \begin{bmatrix} 2x_1 a_1 a_1 + x_2 a_3 a_1 + 2x_1 a_2 a_2 + x_2 a_4 a_2 + x_2 a_1 a_3 + x_2 a_2 a_4 \\ x_1 a_3 a_1 + x_1 a_4 a_2 + x_1 a_1 a_3 + 2x_2 a_3 a_3 + x_1 a_2 a_4 + 2x_2 a_2 a_4 \end{bmatrix}$$

$$= \begin{bmatrix} 2x_1 a_1 a_1 + 2x_2 a_3 a_1 + 2x_2 a_2 a_4 + 2x_1 a_2 a_2 \\ 2x_1 a_3 a_1 + 2x_1 a_4 a_2 + 2x_2 a_3 a_3 + 2x_2 a_4 a_4 \end{bmatrix}$$

$$= 2 \begin{bmatrix} (a_1 a_1 + a_2 a_2) x_1 + (a_1 a_3 + a_2 a_4) x_2 \\ (a_1 a_3 + a_2 a_4) x_1 + (a_3 a_3 + a_4 a_4) x_2 \end{bmatrix}$$

$$= 2 \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Constrained Optimization

where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ and $A = \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix}$

$$\therefore \nabla_x (x^T A^T A x) = 2 \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = 2 A^T A x$$

Constrained Optimization

$$A = \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$A^T A x = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} a_1^2 + a_2^2 & a_1 a_3 + a_2 a_4 \\ a_1 a_3 + a_2 a_4 & a_3^2 + a_4^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$= \begin{bmatrix} (a_1^2 + a_2^2) x_1 + (a_1 a_3 + a_2 a_4) x_2 \\ (a_1 a_3 + a_2 a_4) x_1 + (a_3^2 + a_4^2) x_2 \end{bmatrix} = \begin{bmatrix} F_1 \\ F_2 \end{bmatrix}$$

where $F_1 = (a_1^2 + a_2^2) x_1 + (a_1 a_3 + a_2 a_4) x_2$

$F_2 = (a_1 a_3 + a_2 a_4) x_1 + (a_3^2 + a_4^2) x_2$

Constrained Optimization

$$\nabla_x (A^T A x) = \nabla_x \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} \end{bmatrix}$$

$$= \begin{bmatrix} a_1^2 + a_2^2 & a_1 a_3 + a_2 a_4 \\ a_1 a_3 + a_2 a_4 & a_3^2 + a_4^2 \end{bmatrix}$$

$$= \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} = A^T A$$

Constrained Optimization

- We can then follow this gradient downhill, taking small steps.

Algorithm 4.1 An algorithm to minimize $f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2$ with respect to $\boldsymbol{x}$ using gradient descent, starting from an arbitrary value of $\boldsymbol{x}$.

Set the step size (ϵ) and tolerance (δ) to small, positive numbers.
while $||\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}||_2 > \delta$ **do**
  $\boldsymbol{x} \leftarrow \boldsymbol{x} - \epsilon\left(\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b}\right)$
end while

Constrained Optimization

One can also solve this problem using Newton's method. In this case, because the true function is quadratic, the quadratic approximation employed by Newton's method is exact, and the algorithm converges to the global minimum in a single step.

Now suppose we wish to minimize the same function, but subject to the constraint $\boldsymbol{x}^\top \boldsymbol{x} \leq 1$. To do so, we introduce the Lagrangian

$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda \left(\boldsymbol{x}^\top \boldsymbol{x} - 1 \right).$$

We can now solve the problem

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

Constrained Optimization

- By differentiating the Lagrangian with respect to x, we obtain the equation

$$\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{x} - \boldsymbol{A}^\top \boldsymbol{b} + 2\lambda \boldsymbol{x} = 0.$$

This tells us that the solution will take the form

$$\boldsymbol{x} = (\boldsymbol{A}^\top \boldsymbol{A} + 2\lambda \boldsymbol{I})^{-1} \boldsymbol{A}^\top \boldsymbol{b}.$$

Constrained Optimization

The magnitude of λ must be chosen such that the result obeys the constraint. We can find this value by performing gradient ascent on λ. To do so, observe

$$\frac{\partial}{\partial \lambda} L(\boldsymbol{x}, \lambda) = \boldsymbol{x}^\top \boldsymbol{x} - 1$$

Constrained Optimization

- When the norm of x exceeds 1, this derivative is positive, so to follow the derivative uphill and increase the Lagrangian with respect to λ, we increase λ.
- Because the coefficient on the $x^T x$ penalty has increased, solving the linear equation for x will now yield a solution with smaller norm.
- The process of solving the linear equation and adjusting λ continues until x has the correct norm and the derivative on λ is 0.

Constrained Optimization

$$A = \begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix}$$

$$AA^T = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}\begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} = \begin{bmatrix} a_1a_1 + a_2a_2 & a_1a_3 + a_2a_4 \\ a_1a_3 + a_2a_4 & a_3a_3 + a_4a_4 \end{bmatrix} = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$AA^T + \lambda I = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}\begin{bmatrix} a_1 & a_3 \\ a_2 & a_4 \end{bmatrix} + \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} = \begin{bmatrix} e + \lambda & f \\ g & h + \lambda \end{bmatrix}$$

Constrained Optimization

$$D=\begin{bmatrix} e+\lambda & f \\ g & h+\lambda \end{bmatrix}$$

$$D^{-1}=\begin{bmatrix} e+\lambda & f \\ g & h+\lambda \end{bmatrix} = \frac{1}{((e+\lambda)(h+\lambda)-fg)} \begin{bmatrix} h+\lambda & -f \\ -g & e+\lambda \end{bmatrix}$$

If $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ then

$$A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Inverse of A — Determinant of A — Adjoint of A

Constrained Optimization

$$D^{-1}A^{T}b = \frac{1}{((e+\lambda)(h+\lambda)-fg)}\begin{bmatrix} h+\lambda & -f \\ -g & e+\lambda \end{bmatrix}\begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$D^{-1}A^{T}b = \frac{1}{((e+\lambda)(h+\lambda)-fg)}\begin{bmatrix} h+\lambda & -f \\ -g & e+\lambda \end{bmatrix}\begin{bmatrix} a_1b_1 + a_2b_2 \\ a_3b_1 + a_2b_2 \end{bmatrix}$$

$$D^{-1}A^{T}b = \frac{1}{((e+\lambda)(h+\lambda)-fg)}\begin{bmatrix} h+\lambda & -f \\ -g & e+\lambda \end{bmatrix}\begin{bmatrix} k \\ l \end{bmatrix}$$

$$where\ D^{-1}A^{T}b = (AA^{T}+\lambda I)^{-1}A^{T}b$$

Constrained Optimization

$$D^{-1}A^{T}b = \frac{1}{((e+\lambda)(h+\lambda)-fg)}\begin{bmatrix} hk+\lambda k-fl \\ el+\lambda l-kg \end{bmatrix}$$

$$D^{-1}A^{T}b = \frac{1}{((e+\lambda)(h+\lambda)-fg)}\left[\begin{bmatrix} hk-fl \\ el-kg \end{bmatrix}+\begin{bmatrix} \lambda k \\ \lambda l \end{bmatrix}\right]$$

The End