Real numbers cannot be represented precisely using a finite amount of memory

Infinitely many real numbers but only finite # of bits
This means there is some approximation / rounding error when representing the number in the computer which can become problematic if it accumulates over time

Underflow occurs when small numbers are rounded to zero, which can cause errors like dividing by zero
Overflow occurs when large numbers are approximated as + or - infinite

- Further arithmetic changes these into NaN values

Softmax needs to predict probabilities associated with a multinoulli / categorical distribution while avoiding under / overflow

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

Suppose all members x_i of the distribution have the value c
Then the outputs should be equal to 1/n
However, if c is very negative or positive, this will lead to underflow / overflow

Instead, we should evaluate $\mathrm{softmax}(\boldsymbol{z})$ | $\boldsymbol{z} = \boldsymbol{x} - \max_i x_i$, to avoid underflow / overflow
Similarly, taking the log of softmax must be accounted for
Lower level library developers should know how to deal with underflow / overflow

Conditioning refers to how rapidly a function changes w/ respect to small changes in its inputs
Functions that change rapidly can become problematic since small rounding errors could have large effects

For example, $f(\boldsymbol{x}) = \boldsymbol{A}^{-1}\boldsymbol{x}.$ where A, an n by n matrix, has an eigenvalue decomposition, its condition number is $\max_{i,j} \left| \frac{\lambda_i}{\lambda_j} \right|.$ where the ratio is of the magnitude of the larger value to the smaller one
Matrix inversion is sensitive to error in the input when this number is large
Poorly conditioned matrices amplify errors, causing a compounding effect

Maximization can be accomplished by minimizing -f(x)
The function to be minimized is called the objective function / criterion or the cost function, loss function, or error function

The * superscript indicates the value that minimizes or maximizes a function
$\boldsymbol{x}^* = \arg\min f(\boldsymbol{x}).$

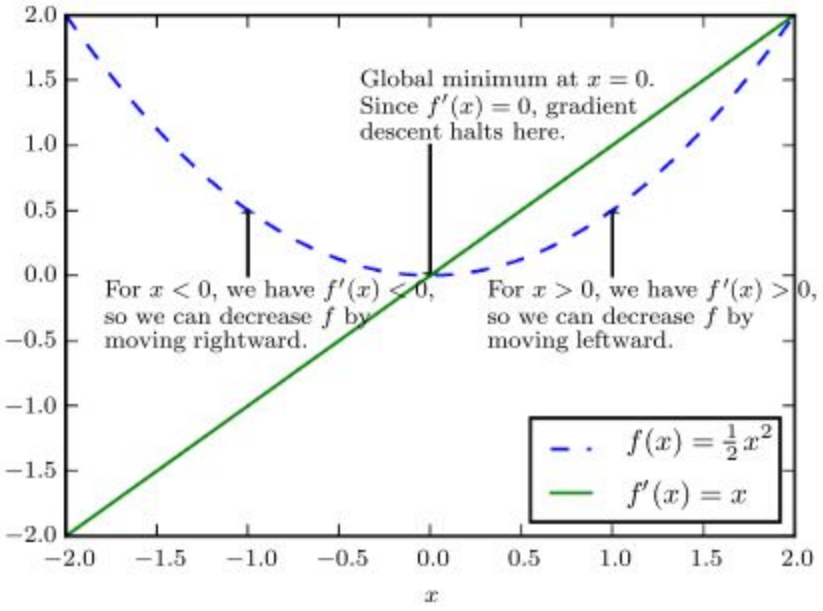


Figure 4.1: An illustration of how the gradient descent algorithm uses the derivatives of a function can be used to follow the function downhill to a minimum.

We can reduce f(x) by moving x in small steps with opposite sign of the derivative (gradient descent)

$f(x - \epsilon \operatorname{sign}(f'(x)))$ is less than $f(x)$

When f'(x) = 0 the derivative does not tell which direction to move
These points are known as critical points or stationary points
You cannot improve the cost function if you are at local minimums or local maximums while making infinitesimal steps
Critical points which are neither minima nor maxima are saddle points

We usually settle for finding a low value for the cost function f since it is unlikely that we will find the true global minimum

For functions with multiple inputs we use partial derivatives $\frac{\partial}{\partial x_i} f(\boldsymbol{x})$ which measures how f changes as only x_i increases at point **x** (vector containing x_{i+1} and other pts)
Gradient generalizes the derivative to the case where the derivative is w/ respect to a vector ; the gradient of f is the vector containing all the partial derivatives

$\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$.

Element i of the gradient is the partial derivative of f with respect to x_i

Critical points in multiple dimensions are where every elements of the gradient = 0
The directional derivative in a direction u (unit vector) is the slope of the function f in the direction u

In other words, it is $\frac{\partial}{\partial \alpha} f(\boldsymbol{x} + \alpha \boldsymbol{u})$ evaluated at alpha = 0

When alpha = 0, the partial derivative evaluates to $\boldsymbol{u}^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$

To minimize f, we want to find the direction in which f decreases the fastest

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

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

where theta is the angle between u and the gradient

Substituting the L2 norm of u as 1, this simplifies to $\min_{\boldsymbol{u}} \cos\theta.$

This is minimized when u points in the opposite direction of the gradient (since they then form a 180 degree angle and their cosine is -1)

The gradient points directly uphill, and the negative gradient points downhill

Decrease f by moving in the direction of the negative gradient => steepest / gradient descent

The new point x' becomes $\boldsymbol{x}' = \boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ where epsilon is the learning rate determining the size of the step

Line search - evalate $f\left(\boldsymbol{x} - \epsilon \nabla_{\boldsymbol{x}} f(\boldsymbol{x})\right)$ for several epsilon values and choose the one with the lowest level

Steepest descent converges when all elements of the gradient = 0 (or close to 0, in practice)

Gradient descent is limited to continuous spaces, but the general idea can be generalized to discrete spaces

Ascending a function of discrete parameters is called hill climbing

Jacobian matrix is used to find the partial derivatives of a function whose input and output are both vectors

$\boldsymbol{f} : \mathbb{R}^m \to \mathbb{R}^n$

$\boldsymbol{J} \in \mathbb{R}^{n \times m}$ of f is defined such that $J_{i,j} = \frac{\partial}{\partial x_j} f(\boldsymbol{x})_i.$

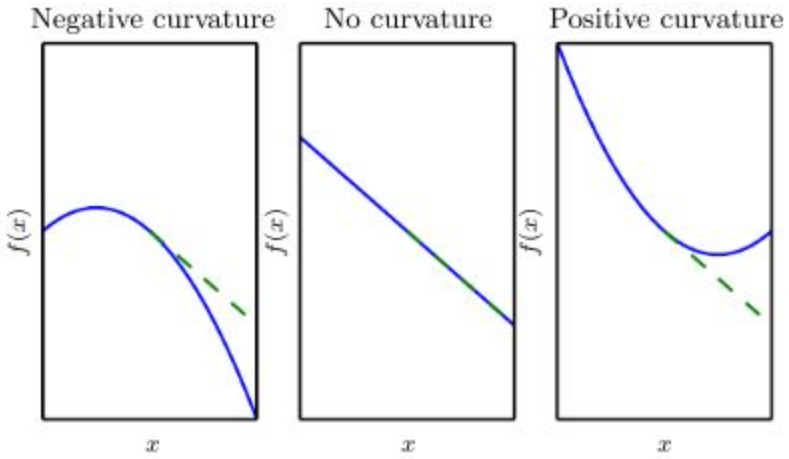
The derivative with respect to x_i of the derivative of f with respect to x_j is denoted as $\frac{\partial^2}{\partial x_i \partial x_j} f.$

The second derivative tells us whether a gradient step will cause as much of an improvement as we would expect based on just the gradient

2nd derivative measures curvature

Based on the curvature measured by the 2nd derivative, we can see how well the gradient predicts

If the second derivative is negative, the function curves downward and the cost function will decrease by more than epsilon



When the function has multiple input dimensions, there exist many second derivatives
These derivatives can be collected into a Hessian matrix H(f)(x) which is defined as

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

The Hessian is the Jacobian of the gradient

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

This property is always true

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

We can decompose Hessians into a set of real eigenvalues and an orthogonal basis of eigenvectors

The second derivative in a specific direction (represented by unit vector d) is given by $\boldsymbol{d}^\top \boldsymbol{H} \boldsymbol{d}.$
When d is an eigenvector of H (or has the same direciton as an eigenvector), 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 the eignevalues with weights between 0 and 1, and eigenvectors that make smaller angle with d receive more weight
The maximum eigenvalue determines the maximum second derivative and the minimum eigenvalue determines the minimum second derivative

The directional second derivative tells us how well we can expect a grad. descent step to perform
We can make a second order Taylor approximation around the point x^0

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

where g is the gradient, H is the hessian at the point

The new point x will be given by x^0 - eg if we use a learning rate of e

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

The expected improvement due to the slope / gradient of the function, the original value of the function, and the correction that must be applied to account for the curvature
When the last term is too large, grad. descent may move uphill
Taylor series is unlikely to remain accurate for a large learning rate, so smaller values of epsilon must be used
When g^THg is positive, the optimal step size that decreases the Taylor approximation of the function (may not be correlated with the function itself, however) is $\epsilon^* = \frac{\boldsymbol{g}^\top \boldsymbol{g}}{\boldsymbol{g}^\top \boldsymbol{H}\boldsymbol{g}}.$

In the worst case when g, the gradient vector, aligns with the eigenvector of H corresponding to the maximal eigenvalue lambda_{max} the optimal step size is given by $1/\text{lambda}_{max}$
The eigenvalues of the Hessian determine the scale of the learning rate

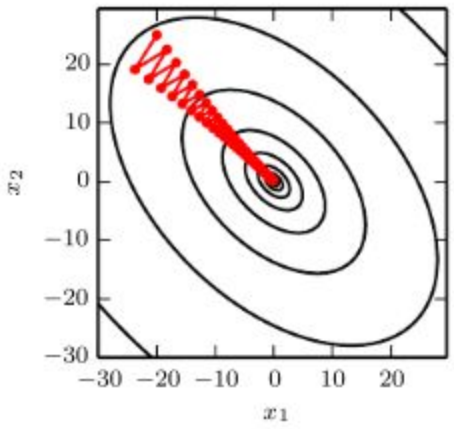
When the second derivative of f is 0, the test is inconclusive ; x may be a saddle point

In multiple dimensions, we examine the second derivatives of the function, generalizing the second derivative test

At a critical point, where $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ = 0, we can examine the eigenvalues of the Hessian to determine whether the critical point is a local min, max, or saddle point
When the Hessian is positive definite (all eigenvalues are > 0) the point is a local minimum since the directional second derivative in any direction must be positive
When at least one eigenvalue is positive and at least one is negative, the point may be a saddle point since it is a local max on one cross section and a local min on another
The test is inconclusive when all non-zero eigenvalues have the same sign, but one eigenvalue is 0

In multiple dimensions, there is a different second derivative for each direction at a single pt
Condition number measures how much second derivatives differ from one another at a particular point
Poor condition number causes gradient descent to perform poorly, since gradient descent does not know which direction to explore

- May choose the one where the derivative decreases rapidly even if there is another option which is better over time
- Also it should not overshoot minimums if there is strong positive curvature



In this case, gradient descent fails to exploit the curvature info contained in the Hessian
The direction of most curvature has 5 times more curvature than least curvature
Gradient descent repeatedly descends canyon walls since they are the steepest features
Since the step size is too large, it overshoots the bottom of the function and spend time descending the opposite canyon wall on the next iteration (rather than going down one side of the hill slowly)
The large positive eigenvalue of the Hessian corresponding to the eigenvector in the shown direction indicates the directional derivative is rapidly increasing

- An optimization algorithm based on the Hessian can predict the steepest direction is not the best search direction

Since minimums may be overshot and the step size must be low, this will cause slow convergence in directions with less curvature

This can be fixed by using info from the Hessian to guide the search
Newton's method is based on using a second order Taylor expansion to approximate f near some point x^0

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

The critical point is

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

When f is a positive definite quadratic function, Newton's method applies the above equation to directly jump to the minimum
When f is not truly quadratic but can be approximated as such, the equation is multiplied multiple times
Iteratively updating the approximation and jumping to the minimum can reach the critical point faster than gradient descent would ; useful near a local minimum, but harmful near saddle point
Gradient descent is not attracted to saddle points unless the gradient points towards them

Optimization algorithms only using gradient, like gradient descent, are first-order optimization algorithms ; second-order optimization algorithms, like Newton's method, use the Hessian

In deep learning, we can gain clarity by constraining the functions to those which are Lipschitz continuous or have Lipschitz continuous derivatives
Lipschitz continuous function is bounded by a lipschitz constant L

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

Quantifies our assumption that a small change in the input made by gradient descent will have a small change in output
Making optimization problems lipschitz continuous can be done w/ minor modifications
Convex optimization, due to different constraints, does not always work in deep learning

Constrained Optimization
Sometimes we want to find the maximum values of f(x) for values of x in a set S
This is constrained optimization
Points x which lie in S are feasible points

Since we want to find a small sol'n, we impose a norm constraint $||\boldsymbol{x}|| \leq 1.$
Can be approached by modifying gradient descent and taking constraints into account
If we have a small constant step size epsilon, we can make gradient descent steps and project the result back onto S ; using a line search, we can search using specific epsilons which yield only feasible new x points
Project gradient into tangent space of feasible region before beginning line search

Another approach is to design a different solution which can be converted to a solution in the original problem
If we want to minimize f(x) for x is a member of R^2 with unit norm, we can instead minimize $g(\theta) = f([\cos\theta, \sin\theta]^\top)$ w/ respect to theta, and return [cos(theta), sin(theta)] as a sol'n to the original problem
Function to minimize depends on the case

Karush-Kuhn-Tucker approach provides a very general sol'n, using a function called the generalized Lagrangian / generalized Lagrange function
We need to describe S in terms of m functions g^i and n functions h^j so that
$\mathbb{S} = \{\boldsymbol{x} \mid \forall i, g^{(i)}(\boldsymbol{x}) = 0 \text{ and } \forall j, h^{(j)}(\boldsymbol{x}) \leq 0\}.$

Equations involving g are the equality constraints and those involving h are called inequality constraints
Lambda and alpha are the KKT multipliers

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

We can solve a constrained minimization problem using unconstrained optimization of the generalized Lagrangian
As long as one feasible point exists and f(x) is not inf, 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 as $\min_{\boldsymbol{x} \in \mathbb{S}} f(\boldsymbol{x}).$ (minimum value of f(x) for each possible point x in set S)

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

This is because any time the constraints are followed
must be true, since lambda does not affect L and alpha makes L smaller than x

but when the constraints are violated

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

Note that this is the generalized lagrangian solution
Max and min notation mean the max / min values of the given variable which satisfy the condition

It is guaranteed that no infeasible point is optimal
To performed constrained maximization, we construct the generalized Lagrange function of -f(x) leading to the optimization problem
Recall maximizing -f(x) minimizes f(x)

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

which can be converted to a problem with maximization in the outer loop

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

The sign of the equality constraints (g and h) do not matter, since the optimization problem will choose any sign for lambdas and alphas

If $h^{(i)}(\boldsymbol{x}^*) = 0.$, then the constraint $h^i(\mathbf{x})$ is active
If a constraint is not active, then the solution to the problem found using the constraint would be at least a local minimum if that constraint was removed
It is possible inactive constraints exclude other solutions
For example, a convex problem with a region of flat, equally globally optimal points, could have a subset of this region eliminaed by constraints
Since an inactive $h^i < 0$, the solution to $\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda}} \max_{\boldsymbol{\alpha}, \boldsymbol{\alpha} \geq 0} L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\alpha})$ will have an alpha of 0 (or L won't be minimized)

Therefore, at the solution $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}.$
For all i, we know that onf of the constraints must be active at the solution
If the solution is on the boundary imposed by the inequality, we must use its KKT multiplier to influence the solution to x ; otherwise, the inequality has no influence on the solution and we represent this by cancelling out its KKT multiplier

KKT conditions are necessary, but not necessarily sufficient, for a point to be optimal

- Gradient of generalized Lagrangian is 0
- All constraints on both x and KKT multipliers are satisfied
- Inequality constraints exhibit complementary slackness $\boldsymbol{\alpha} \odot \boldsymbol{h}(\boldsymbol{x}) = \mathbf{0}.$

We want to find the value of x which minimizes $f(\boldsymbol{x}) = \frac{1}{2}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}||_2^2.$
Use gradient based optimization

The gradient is

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

Set the step size epsilon and tolerance delta 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

Since the true function is quadratic, the quadratic approximation used by Newton's method is exact, and the algorithm converges in a single step

If we want to solve the problem using the constraint $x^Tx <= 1$, introduce the Lagrangian

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

Now, we can solve

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