Multivariate Calculus for ML

Disclaimer: Work in progress. Portions of these written materials are incomplete.

Goals of the course

Intuitive understanding of key differential multivariate calculus concepts

Applications to ML and optimization problems

- Gradient descent, Newton’s method
- Neural Networks and Backpropagation

Practice with Colab + Jax

Assumptions on Audience Background

You have some experience with single variable calculus (Calc I)

- We'll review key concepts before introduction the multivariate analogs

Useful but not crucial to have some basic linear algebra knowledge (vectors, matrices, eigenvalues, invertibility)

Some basic Python experience

Additional resources

- The Matrix Calculus You Need For Deep Learning

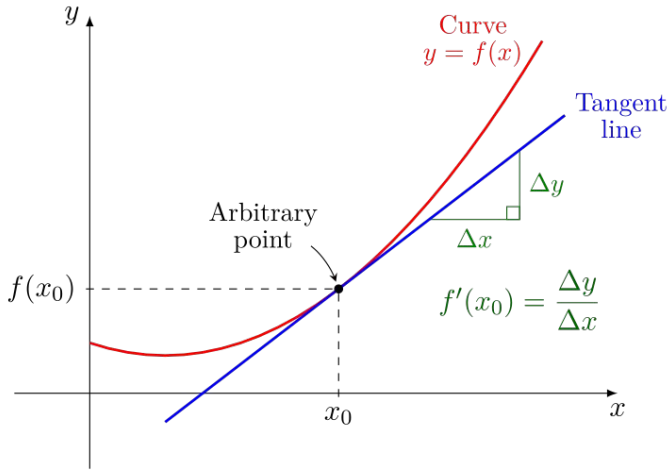
- There are many online calculus textbooks available with many additional examples
  - Early Transcendentals by David Guichard, multivariate material starts at Chapter 14 (page 349)

What is Calculus?

Calculus is the study of change

Two major branches

- **Differential calculus:** derivatives, rates of change, finding extrema, optimization



Calculus is the study of change

Two major branches

- **Differential calculus:** derivatives, rates of change, finding extrema, optimization

- **Integral calculus:** antiderivatives, integration, continuous summation

Calculus is the study of change

Two major branches

- **Differential calculus:** derivatives, rates of change, finding extrema, optimization

  Useful for common ML applications

- **Integral calculus:** antiderivatives, integration, continuous summation

  Useful for theory and proofs, probability and statistics

Calculus and ML

Machine learning is a collection of methods and algorithms to compute functions, using data, often for complex tasks that are difficult to deliberately design.

For example, if we want a function that identifies photos that contain cats, it would be difficult to explicitly write. But given enough data we can find a function – such as a neural network -- and use calculus to fit the function parameters.

In this course we'll learn how this works!

Differential Calculus of single-variable functions

Motivation / History

Calculus was motivated by attempts to understand dynamic motion and other phenomena

In physics, calculus allows one to understand the motion of the planets, velocity, acceleration, electromagnetism, and many other physical quantities

In biology, calculus describes the growth of populations

Countless other examples throughout science, economics, and engineering

Derivatives

Derivatives are the main tool of differential calculus

They allow us to compute the instantaneous rate of change of a function

Derivatives

Derivatives are the main tool of differential calculus

They allow us to compute the instantaneous rate of change of a function

This is the same as the slope of the tangent line of a function at a given point

Derivatives

Derivatives are the main tool of differential calculus

They allow us to compute the instantaneous rate of change of a function

This is the same as the slope of the tangent line of a function at a given point

By computing the slope at every point, we get a new function called the **derivative**

Tangent Lines

If a function $f : \mathbb{R} \to \mathbb{R}$ has a tangent line with slope m at a point $y_0 = f(x_0)$,

the tangent line has the equation:

$$y - y_0 = m(x - x_0)$$

The tangent line is the **best linear approximation** of the function by a line.

Lines are easier to work with and the approximation is good for a small range of values

Derivatives as a function

Given a function, the derivative function is defined to be the slope of the tangent line at each point

$$y = f(x)$$

Two common notations for the derivative function

$f'(x)$ Pronounced “f prime”

$\frac{dy}{dx}$ Pronounced “dee y dee x”

Derivatives as a function

Given a function, the derivative function is defined to be the slope of the tangent line at each point

$$y = f(x)$$

Two common notations for the derivative function

$f'(x)$ Pronounced “f prime”

$\frac{dy}{dx}$ Pronounced “dee y dee x”

Operator notation

$$\frac{d}{dx}(f(x))$$

“Take the derivative with respect to x”

Derivatives of constants and linear functions

- Since a constant function is a horizontal line, the derivative is zero at every point

$$f(x) = c$$

$$f'(x) = 0$$

$$\frac{d}{dx}(c) = 0$$

Derivatives of constants and linear functions

- Since a constant function is a horizontal line, the derivative is zero at every point

$$f(x) = c$$
$$f'(x) = 0$$
$$\frac{d}{dx}(c) = 0$$

- Similarly, for a line of slope m the derivative is the constant function with value m

$$f(x) = mx + b$$
$$f'(x) = m$$
$$\frac{d}{dx}(mx + b) = m$$

Derivative rules

There are many other rules, but it's not necessary to remember them all

Can look them up or use software to compute symbolic derivatives

And now we have autodiff tools in libraries like Jax

$$\frac{d}{dx}e^x = e^x$$

$$\frac{d}{dx}\log x = \frac{1}{x}$$

$$\frac{d}{dx}\sin(x) = \cos(x)$$

$$\frac{d}{dx}\cos(x) = -\sin(x)$$

Derivative rules

Good to be familiar with a few of the higher level rules

Power rule $$\frac{d}{dx}x^n = nx^{n-1}$$

Sum rule $$\frac{d}{dx}(f+g) = \frac{df}{dx} + \frac{dg}{dx}$$

Product rule $$\frac{d}{dx}(f*g) = \frac{df}{dx}*g + f*\frac{dg}{dx}$$

For constants c: $$\frac{d}{dx}(cf) = c\frac{df}{dx}$$

Derivative rules: Chain rule

Derivative of composition of functions

$$(f \circ g)(x) = f(g(x))$$

$$(f \circ g)' = (f' \circ g)(x)g'(x) = f'(g(x))g'(x)$$

$$\frac{d}{dx}(f \circ g) = \frac{df}{dg}\frac{dg}{dx}$$

Derivative example (ML)

Activation functions in neural networks

$$f(x) = \frac{1}{1 + e^{-x}}$$

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

Convenient computationally since we don't have to compute another function to compute the derivative

TODO

Higher order derivatives

We can iterate the process of differentiation to get higher order derivatives

Second derivative

$$f''(x) = (f'(x))'$$

$$\frac{d^2 f}{dx^2} = \frac{d}{dx}\left(\frac{df}{dx}\right)$$

n-th derivative

$$f^{(n)}(x)$$

$$\frac{d^n f}{dx^n}$$

TODO

Higher order derivative examples

$$
\begin{aligned}
f(x) &= x^3 + 4x^2 - 5x + 1 \\
f'(x) &= 3x^2 + 8x - 5 \\
f''(x) &= 6x + 8 \\
f'''(x) &= 6 \\
f^{(4)}(x) &= 0
\end{aligned}
$$

$$
\frac{d^n}{dx^n} e^x = e^x
$$

Examples of non-differentiable functions

Not all functions are differentiable – any point that has a corner, spike, or gap does not have a unique tangent line

A function can fail to have higher derivatives

Single variable optimization

Optimization with derivatives

Very commonly we want to find maxima or minima of functions:

- To maximize profit or growth
- To minimize route lengths, energy consumption, latency
- To minimize total error when fitting ML or statistical models, or to maximize some quantity like accuracy

Local Extrema

For a differentiable function, the derivative is zero at local maxima and minima

Local Extrema

For a differentiable function, the derivative is zero at local maxima and minima

Geometrically, the tangent line is horizontal at local extrema and inflection points

Local Extrema

For a differentiable function, the derivative is zero at local maxima and minima

Geometrically, the tangent line is horizontal at local extrema and inflection points

Global extrema not necessarily local extrema

Local Extrema: algebraic example

Values where the derivative is zero (or undefined) are called **critical points**

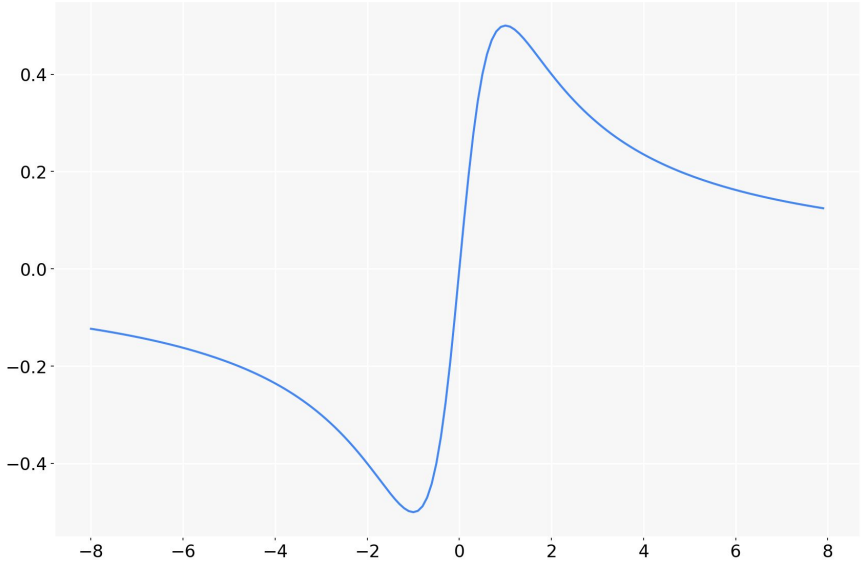
$$f'(c) = 0$$

Local extreme values occur at these points.

Sometimes we can explicitly solve for them.

Local Extrema: algebraic example

$$f(x) = \frac{x}{x^2 + 1}$$



Local Extrema: algebraic example

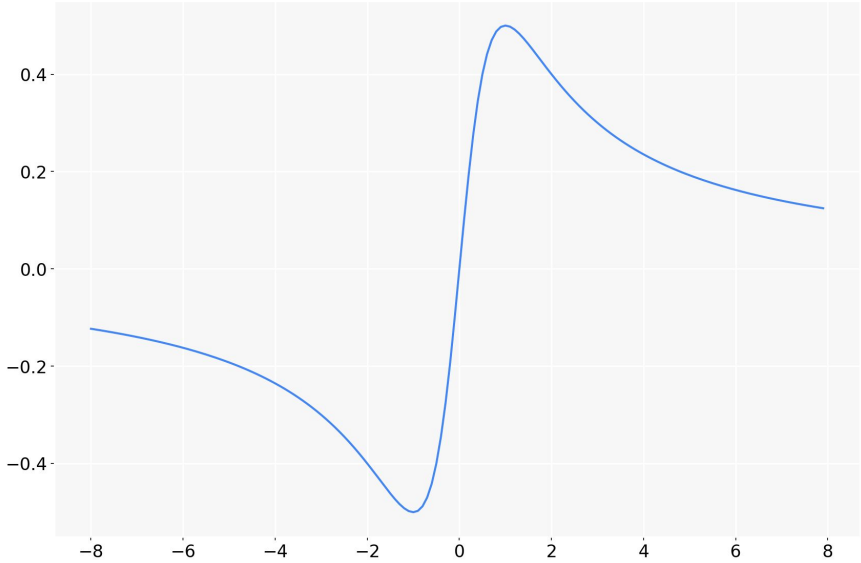
$$f(x) = \frac{x}{x^2 + 1}$$

Derivative:

$$\frac{df}{dx} = \frac{1 - x^2}{(x^2 + 1)^2} = 0$$

Solve for critical points:

$$x = -1, 1$$



How do we distinguish minima and maxima?

Second derivatives!

- If the second derivative is **positive**, the critical point is a **local minimum**
- If the second derivative is **negative**, the critical point is a **local maximum**

$$f''(x) > 0$$

$$f''(x) < 0$$

Otherwise we can look at how the first derivative changes

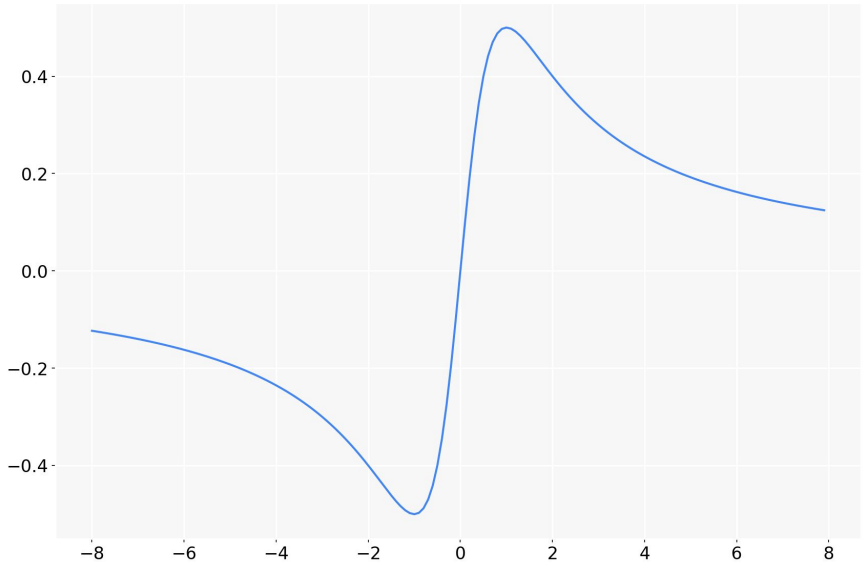
- If the first derivative goes from positive to negative, it's a **local maximum**
- If the first derivative goes from negative to positive, it's a **local minimum**

Local Extrema: example (cont.)

$$f(x) = \frac{x}{x^2+1}$$

Critical points

$$\frac{df}{dx} = \frac{1-x^2}{(x^2+1)^2} = 0 \qquad x = -1, 1$$



Local Extrema: example (cont.)

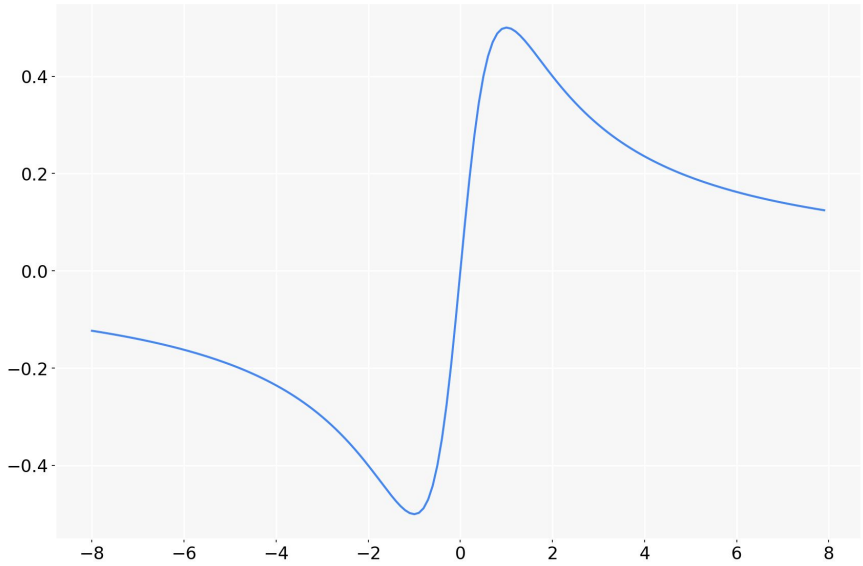
$$f(x) = \frac{x}{x^2+1}$$

Critical points

$$\frac{df}{dx} = \frac{1-x^2}{(x^2+1)^2} = 0 \qquad x = -1, 1$$

Second Derivative

$$\frac{d^2f}{dx^2} = \frac{2x(x^2-3)}{(x^2+1)^3}$$



Local Extrema: example (cont.)

$$f(x) = \frac{x}{x^2+1}$$

Critical points

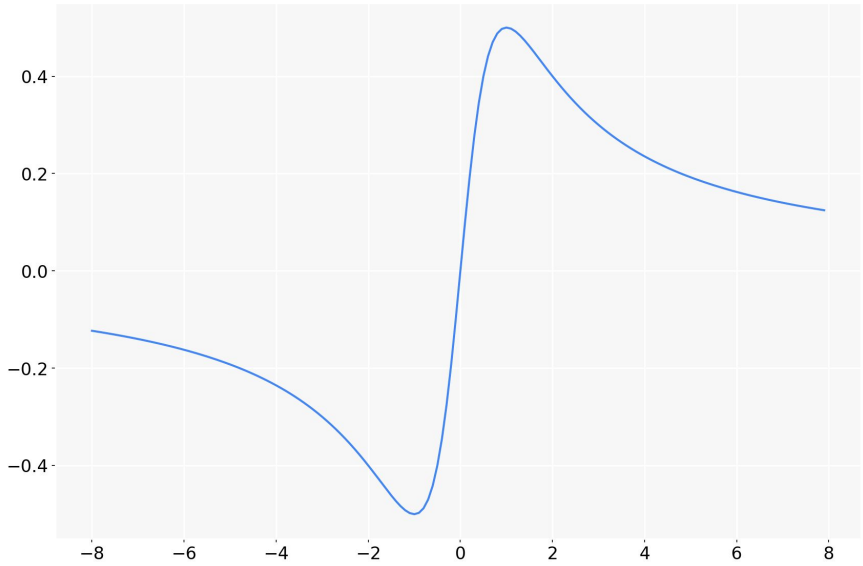
$$\frac{df}{dx} = \frac{1-x^2}{(x^2+1)^2} = 0 \qquad x = -1, 1$$

Second Derivative

$$\frac{d^2f}{dx^2} = \frac{2x(x^2-3)}{(x^2+1)^3}$$

$$\frac{d^2f}{dx^2}(1) = -\frac{1}{2} < 0 \quad \textbf{Local Max}$$

$$\frac{d^2f}{dx^2}(-1) = \frac{1}{2} > 0 \quad \textbf{Local Min}$$



Inflection points

Critical points don't have to be a local maximum or minimum. They can also be inflection points.

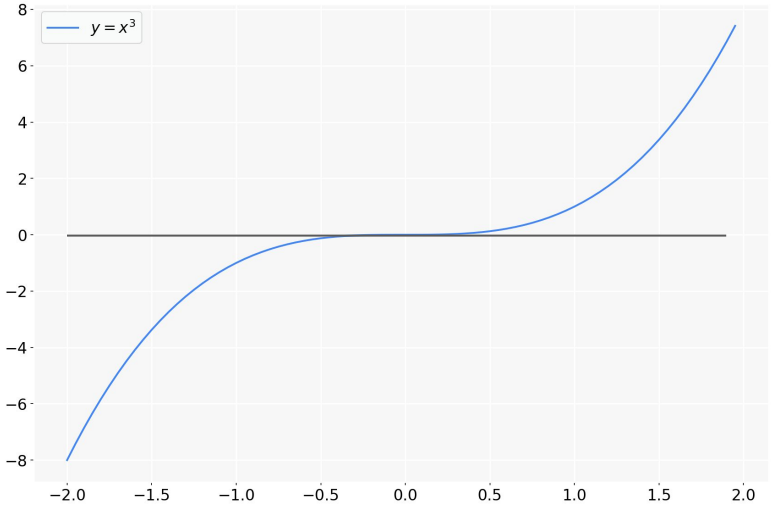
Notice that the tangent line at the origin is horizontal.

$$\frac{dy}{dx} = 3x^2 = 0 \qquad \text{at} \qquad x = 0$$

$$\frac{d^2y}{dx^2} = 6x = 0 \qquad \text{at} \qquad x = 0$$

Second derivative neither positive nor negative

$$y = x^3$$



Gradient Descent

Local Extrema: gradient descent

Often too difficult or impossible to solve for critical points algebraically

Instead we use an iterative approach called **gradient descent** to find local minima

Local Extrema: gradient descent

Basic algorithm:

- Start with an initial guess

- Follow the direction of steepest descent to a new value
  - In one dimension, this is the negative of the derivative

- Continue until we reach a local minimum, where the derivative is zero

Local Extrema: gradient descent

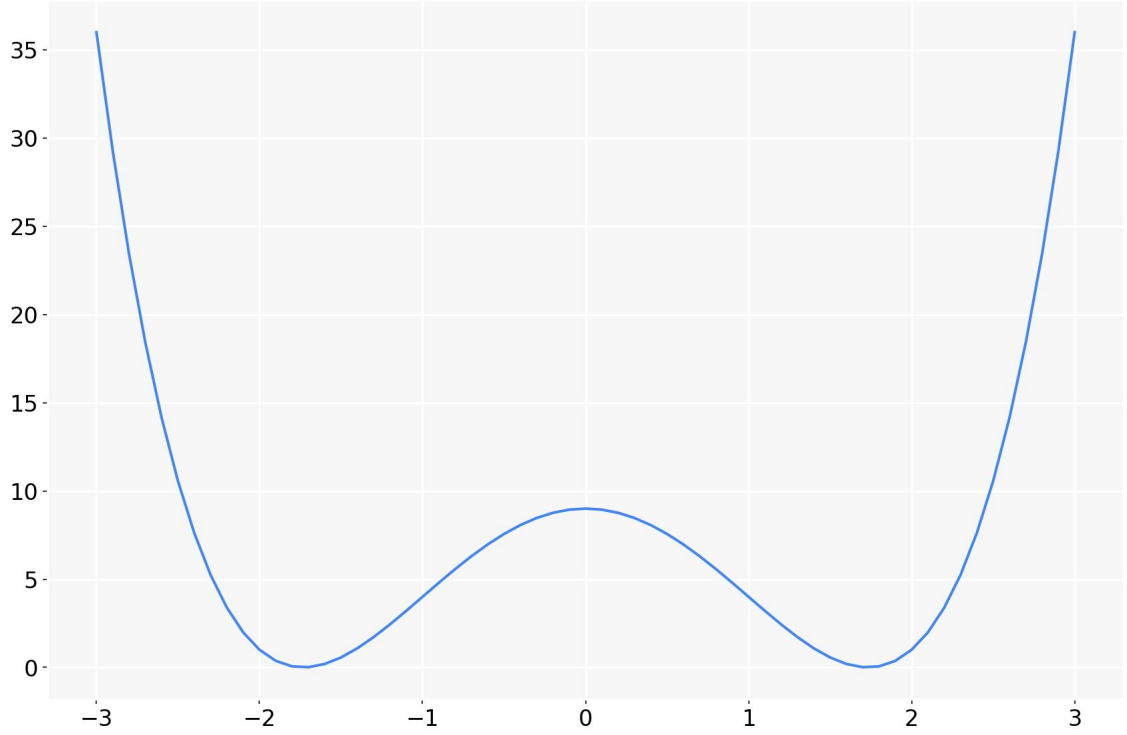
- Start with an initial x_0
- Choose a positive learning rate α
- Compute iterates according to:

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

Gradient descent: numerical example

$$f(x) = \left(x^2 - 3\right)^2$$

Two minima at $\pm\sqrt{3}$



Gradient descent: numerical example

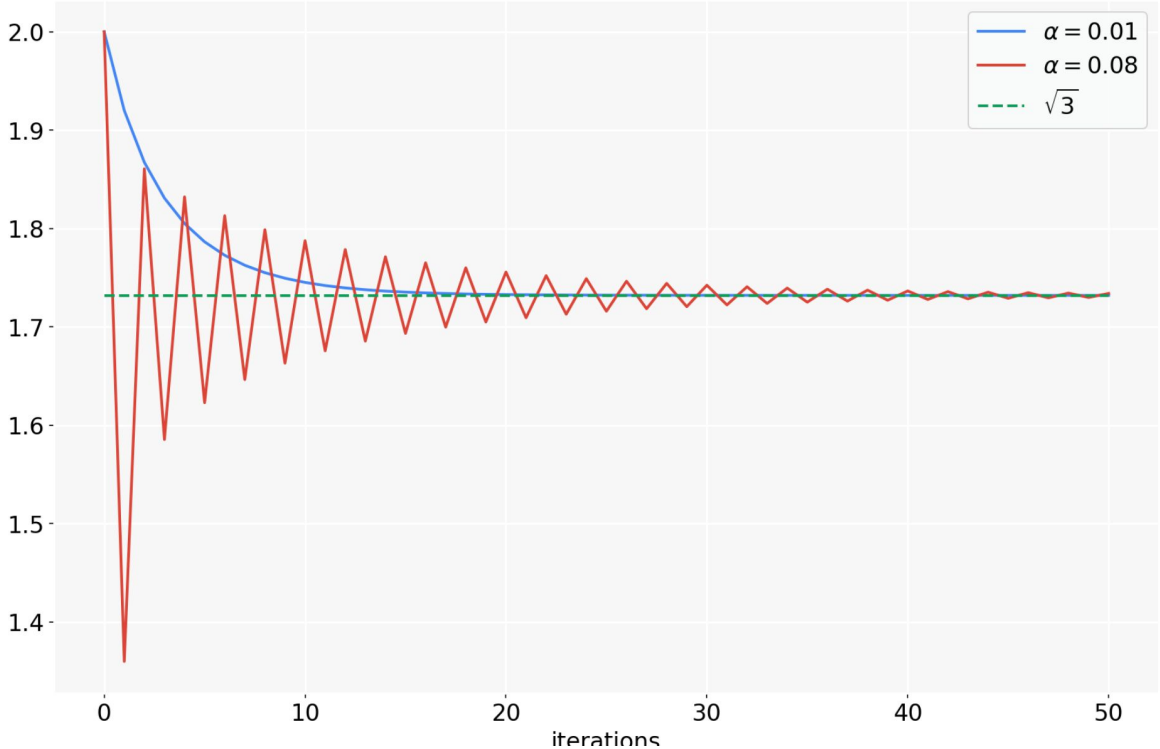
$$f(x) = \left(x^2 - 3\right)^2$$

Two minima at $\pm\sqrt{3}$

Using an initial point of 2, gradient descent converges to the positive minimum.

Note the quite different paths depending on the learning rate

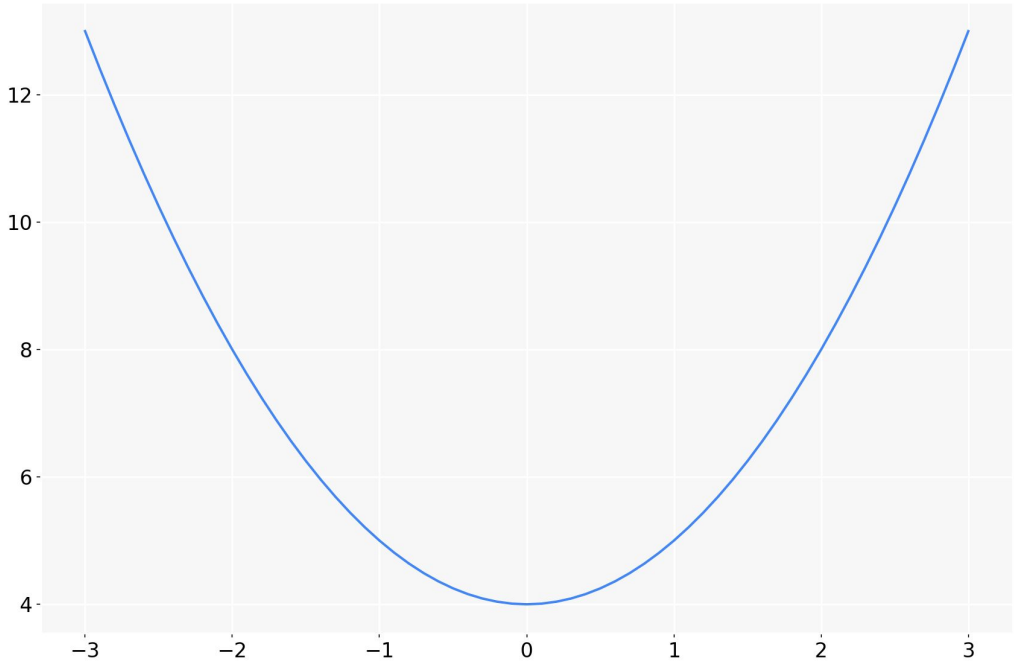
Smaller initial values may converge to the negative minimum



Gradient descent: example with proof of convergence

$$f(x) = x^2 + 4$$

$$f'(x) = 2x$$



Minimum at zero

Gradient descent: example with proof of convergence

$$f(x) = x^2 + 4$$

$$f'(x) = 2x$$

Gradient Descent

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

$$x_{n+1} = x_n - \alpha(2x_n) = x_n(1 - 2\alpha)$$

$$x_m = x_0(1 - 2\alpha)^m$$

Gradient descent: example with proof of convergence

$$f(x) = x^2 + 4$$

$$f'(x) = 2x$$

When $\alpha < 1/2$ ther $x_m \to 0$

which is the minimum of f(x)

Gradient Descent

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

$$x_{n+1} = x_n - \alpha(2x_n) = x_n(1 - 2\alpha)$$

$$x_m = x_0(1 - 2\alpha)^m$$

ML example: Linear Regression

Given a set of data of the form (x_i, y_i)

we want to find a best fit line $y = mx$

We can do this by minimizing the sum of squared errors with gradient descent

$$S = \sum_i (y_i - mx_i)^2$$

$$\frac{dS}{dm} = \sum_i 2(y_i - mx_i)x_i$$

Newton’s Method

There’s another optimization method used to find zeros of functions, called Newton’s method

The idea is to use the tangent line to approximate the location of a zero, iterating until convergence

Iteration is simply finding were successive tangent lines intercept the x-axis (have a zero)

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

Newton's Method

Main idea: take the tangent line at x_n and choose x_{n+1} so that $y = 0$

$$y = f'(x_n)\,(x - x_n) + f(x_n)$$

$$0 = f'(x_n)\,(x_{n+1} - x_n) + f(x_n)$$

Then rearrange:

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

Newton's Method for optimization

We can also use Newton's method to find local extrema by applying the **method to the derivative**

This will find **critical points** of our function (zeros of the derivative) instead of zeros of the original function

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

Newton's Method for optimization

We can also use Newton's method to find local extrema by applying the **method to the derivative**

This will find **critical points** of our function (zeros of the derivative) instead of zeros of the original function

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

We can also add a learning rate as in gradient descent to control convergence behavior

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

Comparing Gradient Descent and Newton's Method

Gradient Descent (first order)

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

Newton's Method (second order)

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

Newton's method uses information from the second derivative to *condition* gradient (first derivative)

This can speed up convergence but also cause issues if the second derivative is ever zero, doesn't exist, or is difficult/expensive to compute

Differential Calculus of multivariate functions

Derivatives of Multivariate functions

Consider a two variable scalar function $z = f(x, y)$

Algebraically there should be at least two derivatives, one for each variable

Derivatives of Multivariate functions

Consider a two variable scalar function $z = f(x, y)$

Algebraically there should be at least two derivatives, one for each variable

Geometrically, we have a tangent plane instead of a tangent line

And we can ask about the rate of change in many directions

Directional derivatives

For a single variable function, there's only "one" direction to move – along the tangent line (forward or backward)

In two dimensions, we can move in the direction of any unit vector in the tangent plane, so there are infinitely many **directional derivatives**

Partial derivatives

Thankfully, we need only compute two special derivatives, one for each variable, and then we can compute any directional derivative from those

These are called **partial derivatives.** We compute them by holding all the other variables constant.

$$z = f(x, y)$$

$$\frac{\partial f}{\partial x}$$

$$\frac{\partial f}{\partial y}$$

Partial derivatives: example

$$f(x, y) = x^2 + y^3$$

$$\frac{\partial f}{\partial x} = 2x$$

For the partial with respect to variable x, variable y is considered constant, so all terms with only y disappear

Partial derivatives: example

$$f(x, y) = x^2 + y^3$$

$$\frac{\partial f}{\partial x} = 2x$$

For the partial with respect to variable x, variable y is considered constant, so all terms with only y disappear

$$\frac{\partial f}{\partial y} = 3y^2$$

For the partial with respect to variable y, variable x is considered constant, so all terms with only x disappear

Partial derivatives: Second derivatives

There are four second derivatives now, but the order usually doesn’t matter, so some are equal

$$f(x, y) = x^2 y$$

Partial derivatives: Second derivatives

There are four second derivatives now, but the order usually doesn't matter, so some are equal

$$f(x,y) = x^2 y$$

First derivatives

$$\frac{\partial f}{\partial x} = 2xy$$
$$\frac{\partial f}{\partial y} = x^2$$

Partial derivatives: Second derivatives

There are four second derivatives now, but the order usually doesn't matter, so some are equal

$$f(x,y) = x^2 y$$

First derivatives

$$\frac{\partial f}{\partial x} = 2xy$$
$$\frac{\partial f}{\partial y} = x^2$$

Second derivatives – take partial derivatives of first derivatives

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial f}{\partial x}\right) = 2y$$
$$\frac{\partial^2 f}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial f}{\partial y}\right) = 0$$
$$\frac{\partial^2 f}{\partial x \partial y} = 2x = \frac{\partial^2 f}{\partial y \partial x}$$

Gradients

$$y = f(x_1, x_2, \ldots, x_n)$$

The (n x 1) column vector of partial first derivatives is called the **gradient**

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \ldots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right]^T$$

Gradients

$$y = f(x_1, x_2, \ldots, x_n)$$

The (n x 1) column vector of partial first derivatives is called the **gradient**

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \ldots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

For a single variable, the gradient is just the usual derivative as a 1x1 vector

$$y = f(x)$$

$$\nabla f = \left[\frac{df}{dx}\right]$$

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]^T$$

Gradient example (entropy)

Discrete probability distribution

$$x_1 + x_2 + \ldots + x_n = 1$$
$$\mathbf{x} = (x_1, \cdots, x_n)$$

Shannon entropy

$$H(\mathbf{x}) = -\sum_i x_i \log x_i$$
$$\frac{\partial H(\mathbf{x})}{\partial x_i} = -\log x_i - 1$$

Gradient example (entropy)

Discrete probability distribution

$$x_1 + x_2 + \ldots + x_n = 1$$
$$\mathbf{x} = (x_1, \cdots, x_n)$$

Shannon entropy

$$H(\mathbf{x}) = -\sum_i x_i \log x_i$$
$$\frac{\partial H(\mathbf{x})}{\partial x_i} = -\log x_i - 1$$

Gradient

$$\nabla H = \begin{bmatrix} -\log x_1 - 1 \\ \cdots \\ -\log x_n - 1 \end{bmatrix}$$
$$= -\begin{bmatrix} \log x_1 \\ \cdots \\ \log x_n \end{bmatrix} - \begin{bmatrix} 1 \\ \cdots \\ 1 \end{bmatrix}$$

Directional Derivative and Gradients

With the gradient, we can compute the directional derivative in the direction of any unit vector with a dot product

$$D_u(f) = \nabla f \cdot u$$

Directional Derivative and Gradients

With the gradient, we can compute the directional derivative in the direction of any unit vector with a dot product

$$D_u(f) = \nabla f \cdot u$$

Partial derivatives correspond to unit vectors concentrated in one dimension

$$u = (1, 0)$$
$$D_u(f) = \nabla f \cdot u = \frac{\partial f}{\partial x}$$

$$v = (0, 1)$$
$$D_v(f) = \nabla f \cdot v = \frac{\partial f}{\partial y}$$

Gradients: steepest ascent

The gradient is special – it's the direction of steepest ascent

$$||\nabla f \cdot u|| = ||\nabla f||||u|| \cos\theta$$

Largest when angle between is zero!

The directional derivative is greatest when the direction is the same as the gradient

So for optimization we typically use the **gradient** as our derivative

Derivative of vector-valued functions

$$f(x_1, x_2, \ldots, x_n) = [f_1, \ldots, f_m]^T$$

For a vector-valued function with **n** variables and **m** outputs, also called a **vector field**, there are **m x n** first partial derivatives

This is called the **Jacobian** matrix

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Derivative of vector-valued functions

Example:

$$f(x_1, \ldots, x_n) = [x_1, \ldots, x_n]^T$$

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

The Jacobian is the identity matrix

$$\frac{\partial y_i}{\partial x_j} = \delta_{ij} = 1 \text{ if } i = j, \text{ else } 0$$

Derivative of vector-valued functions

More generally, for a function given by matrix multiplication:

$$f(\mathbf{x}) = A\mathbf{x}$$

$$J(f) = A$$

The previous example was for A = I the identity matrix

Summary: Derivative, Gradient, Jacobian

Function type	Variables	Outputs	Derivatives
Scalar of 1 variable	1	1	1 ordinary derivative
Scalar of n variables	n	1	n partial derivatives (gradient)
Vector-valued of n-variables and m-outputs (vector field)	n	m	m x n partial derivatives (Jacobian)

$$\frac{df}{dx}$$

$$\frac{\partial f}{\partial x_i}$$

$$\frac{\partial f_j}{\partial x_i}$$

Multivariate Optimization

Multidimensional critical points

For a scalar function of a single variable, we looked for points where the **derivative was zero or undefined**

For a scalar function of many variables, we want **each partial derivative to be zero or undefined**

$$\frac{\partial f}{\partial x_i} = 0$$

This is the same as the gradient being the zero vector or undefined

$$\nabla f = \mathbf{0}$$

Multidimensional critical points

We still can have **local maxima** and **local minima** at critical points

Multidimensional inflection points are called **saddle points**

Sphere

The upper hemisphere of the unit sphere is given by the function

$$z = f(x, y) = \sqrt{1 - x^2 - y^2}$$

Geometrically, we have a **local max** at the north pole,
where the **tangent plane is horizontal**

Sphere

The upper hemisphere of the unit sphere is given by the function

$$z = f(x, y) = \sqrt{1 - x^2 - y^2}$$

Geometrically, we have a **local max** at the north pole, where the **tangent plane is horizontal**

The critical point occurs where both partials are zero, which is the origin (0, 0)

$$\frac{\partial f}{\partial x} = \frac{-2x}{\sqrt{1 - x^2 - y^2}} = 0 \qquad \Rightarrow x = 0$$

Max, Min, or Saddle?

For single variable functions, we looked at the second derivative:

- If the second derivative is **positive** at the critical point, we have a **local min**
- If the second derivative is **negative** at the critical point, we have a **local max**
- If the second derivative is **zero** at the critical point, we could have any of the three (need to look at how the first derivative changes)

For multiple variables we can do something similar, but recall that there are many second derivatives

Max, Min, or Saddle?

For single variable functions, we looked at the second derivative:

- If the second derivative is **positive** at the critical point, we have a **local min**
- If the second derivative is **negative** at the critical point, we have a **local max**
- If the second derivative is **zero** at the critical point, we could have any of the three (need to look at how the first derivative changes)

For multiple variables we can do something similar, but recall that there are many second derivatives

So first let's understand the second derivatives of scalar functions of many variables

Hessian: Multivariate second derivative of scalar functions

- First derivative of scalar function is the gradient, which is a vector-valued function

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

Hessian: Multivariate second derivative of scalar functions

- First derivative of scalar function is the gradient, which is a vector-valued function

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

- Second derivative is then the Jacobian of the gradient, called the **Hessian**, consisting of all the second derivatives

$$H(f(\mathbf{x})) = J(\nabla f(\mathbf{x}))$$

Hessian: Multivariate second derivative of scalar functions

- First derivative of scalar function is the gradient, which is a vector-valued function

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \cdots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

- Second derivative is then the Jacobian of the gradient, called the **Hessian**, consisting of all the second derivatives

$$H(f(\mathbf{x})) = J(\nabla f(\mathbf{x}))$$

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1\, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1\, \partial x_n} \\ \frac{\partial^2 f}{\partial x_2\, \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2\, \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n\, \partial x_1} & \frac{\partial^2 f}{\partial x_n\, \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- The Hessian is usually a **symmetric matrix** (order of partial derivatives doesn't matter for nice functions

Hessian: Examples (Scalar-valued multivariate functions)

Gradient and Hessian of summation

$$f(\mathbf{x}) = x_1 + \cdots + x_n$$

$$\nabla f = [1, \ldots, 1]$$

$$H(f) = J(\nabla f) = 0$$

Hessian: Examples (Scalar-valued multivariate functions)

Gradient and Hessian of summation

$$f(\mathbf{x}) = x_1 + \cdots + x_n$$
$$\nabla f = [1, \ldots, 1]$$
$$H(f) = J(\nabla f) = 0$$

Quadratic form given by a matrix A

$$f(\mathbf{x}) = \mathbf{x} \cdot A\mathbf{x}$$
$$\nabla f = (A + A^T)\mathbf{x}$$
$$H(f) = J(\nabla f) = A + A^T$$

Hessian: Least Squares

$$f(\mathbf{x}) = \frac{1}{2}||A\mathbf{x} - \mathbf{y}||^2$$
$$= \frac{1}{2}\mathbf{x}^T(A^TA)\mathbf{x} - \mathbf{x}^TA^T\mathbf{y} - \frac{1}{2}\mathbf{y}^T\mathbf{y}$$

$$\nabla f = (A^TA)\mathbf{x} - A^T\mathbf{y}$$
$$H(f) = A^TA$$

Compare to:

$$\frac{d}{dx}(mx + b) = m$$

Multivariate Extrema

To determine what kind of extrema a critical point may be, we look at the Hessian matrix **H** of second derivatives:

- If **H** is positive definite, the critical point is a local minimum

$$M \text{ positive-definite} \iff \mathbf{x}^{\mathsf{T}} M \mathbf{x} > 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

- If **H** is negative definite, the critical point is a local maximum

$$M \text{ negative-definite} \iff \mathbf{x}^{\mathsf{T}} M \mathbf{x} < 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

- If **H** is indefinite, the critical point is a saddle point

Multivariate Extrema

To determine what kind of extrema a critical point may be, we look at the Hessian matrix **H** of second derivatives:

- If **H** is positive definite, the critical point is a local minimum

$$M \text{ positive-definite} \iff \mathbf{x}^{\mathsf{T}} M\mathbf{x} > 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

- If **H** is negative definite, the critical point is a local maximum

$$M \text{ negative-definite} \iff \mathbf{x}^{\mathsf{T}} M\mathbf{x} < 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

- If **H** is indefinite, the critical point is a saddle point

Single variable

$$f''(x) > 0$$

$$f''(x) < 0$$

Multivariate Extrema

To determine what kind of extrema a critical point may be, we look at the Hessian matrix **H** of second derivatives:

- If **H** is positive definite, the critical point is a local minimum

$$M \text{ positive-definite} \iff \mathbf{x}^{\mathsf{T}} M\mathbf{x} > 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

- If **H** is negative definite, the critical point is a local maximum

$$M \text{ negative-definite} \iff \mathbf{x}^{\mathsf{T}} M\mathbf{x} < 0 \text{ for all } \mathbf{x} \in \mathbb{R}^n \setminus \{\mathbf{0}\}$$

- If **H** is indefinite, the critical point is a saddle point

Eigenvalues of M

All positive

All negative

Some positive, some negative

Gradient Descent: Multivariate

Multivariate gradient descent works essentially like the single variable case, where the gradient plays the role of the derivative

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

We iterate until until we hit a local minimum, where the gradient is zero.

Gradient Descent: Multivariate

$$z = f(x, y) = x^2 - y^2$$

Initial point: x = -0.7, y = 0.0

Initial point: x = -0.7, y = -0.01

Newton's Method: multivariate

Similarly, for the multivariate generalization of Newton's method, we use the Gradient and Hessian as the first and second derivatives

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

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \alpha H(\mathbf{x_n})^{-1} \nabla f(\mathbf{x_n})$$

Note that we have to take the matrix inverse of the Hessian to multiply with the gradient

End of Session 1

Loss functions

p-norm:

$$||x||_p = \left(\sum_i |x_i|^p \right)^{\frac{1}{p}}$$

Mean Absolute Error (p=1)

$$L_1 = \frac{1}{n} \sum_i |y_i - f(x_i)|$$

For ***n*** data points (x_i, y_i)

Squared errors (p=1)

$$L_2 = \frac{1}{n} \sum_i |y_i - f(x_i)|^2$$

Modeled by $y = f(x_i) + \epsilon$

We can define loss functions $L_p = \frac{1}{n} ||y - f(x)||_p^p$

Calculus and ML

Machine learning is a collection of methods and algorithms to compute functions, using data, often for complex tasks that are difficult to deliberately design.

For example, if we want a function that identifies photos that contain cats, it would be difficult to explicitly write. But given enough data we can find a function – such as a neural network -- and use calculus to fit the function parameters.

In this course we'll learn how this works!

Gradients

$$y = f(x_1, x_2, \ldots, x_n)$$

The (n x 1) column vector of partial first derivatives is called the **gradient**

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \ldots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right]^T$$

Gradients

$$y = f(x_1, x_2, \ldots, x_n)$$

The (n x 1) column vector of partial first derivatives is called the **gradient**

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \ldots \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

For a single variable, the gradient is just the usual derivative as a 1x1 vector

$$y = f(x)$$

$$\nabla f = \left[\frac{df}{dx}\right]$$

$$\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]^T$$

Gradients: steepest ascent

The gradient is special – it's the direction of steepest ascent

$$||\nabla f \cdot u|| = ||\nabla f||||u|| \cos\theta$$

Largest when angle between is zero!

The directional derivative is greatest when the direction is the same as the gradient

Local Extrema: gradient descent

Basic algorithm:

- Start with an initial guess

- Follow the direction of steepest descent to a new value
  - In one dimension, this is the negative of the derivative

- Continue until we reach a local minimum, where the derivative is zero

Gradient Descent: Multivariate

Multivariate gradient descent works essentially like the single variable case, where the gradient plays the role of the derivative

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

We iterate until until we hit a local minimum, where the gradient is zero.

Jacobian: Derivative of vector-valued functions

$$f(x_1, x_2, \ldots, x_n) = [f_1, \ldots, f_m]^T$$

For a vector-valued function with **n** variables and **m** outputs, also called a **vector field**, there are **m x n** first partial derivatives

This is called the **Jacobian** matrix

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \nabla^{\mathrm{T}} f_1 \\ \vdots \\ \nabla^{\mathrm{T}} f_m \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Summary: Derivative, Gradient, Jacobian

Function type	Variables	Outputs	Derivatives
Scalar of 1 variable	1	1	1 ordinary derivative
Scalar of n variables	n	1	n partial derivatives (gradient)
Vector-valued of n-variables and m-outputs (vector field)	n	m	m x n partial derivatives (Jacobian)

$$\frac{df}{dx}$$

$$\frac{\partial f}{\partial x_i}$$

$$\frac{\partial f_j}{\partial x_i}$$

Motivation: Neural Networks

Feed Forward Neural Networks

A single layer (feed forward) NN is essentially a function of the form

$$g(\mathbf{x}) = f(W\mathbf{x} + \mathbf{b})$$

Where **f** is an activation function and **W** is a matrix of weights between nodes in inner layer

Feed Forward Neural Networks

A multilayer (feed forward) NN is a composition of such functions

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots))$$

The exponents are “upper indices” indicating the layer, not powers or iterates of a function or matrix

Feed Forward Neural Networks

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots))$$

Key points:

- A neural network isn't an arbitrary function of many variables, rather it's structured in layers
- Each layer can be viewed as a function that depends on the parameters from the prior layer(s)

Fitting Neural Networks

To fit a neural network, we perform gradient descent on some loss function composed with g(x).

Fitting Neural Networks

To fit a neural network, we perform gradient descent on some loss function composed with g(x).

Given a set of data $(\mathbf{x}^i, \mathbf{y}^i)$

We need to minimize a loss function $C = \frac{1}{n} \sum_i \left(\mathbf{y}^i - g(\mathbf{x}^i)\right)^2$

where

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots))$$

Fitting Neural Networks

To fit a neural network, we perform gradient descent on some loss function composed with g(x).

Given a set of data $(\mathbf{x}^i, \mathbf{y}^i)$

We need to minimize a loss function $C = \frac{1}{n} \sum_i \left(\mathbf{y}^i - g(\mathbf{x}^i)\right)^2$

where

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots))$$

Fitting Neural Networks

To fit a neural network, we perform gradient descent on some loss function composed with g(x).

Given a set of data $(\mathbf{x}^i, \mathbf{y}^i)$

We need to minimize a loss function $C = \frac{1}{n} \sum_i \left(\mathbf{y}^i - g(\mathbf{x}^i)\right)^2$

where

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots))$$

Fitting Neural Networks

To fit a neural network, we perform gradient descent on some loss function composed with g(x).

Given a set of data $(\mathbf{x}^i, \mathbf{y}^i)$

We need to minimize a loss function $C = \frac{1}{n} \sum_i \left(\mathbf{y}^i - g(\mathbf{x}^i)\right)^2$

where

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots))$$

To understand how this works we need multivariate chain rules

Multivariate Chain Rules

Chain rule: single variable scalar function

$$y = f(x) = f(g(t))$$

$$x = g(t)$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

Chain rule: single variable scalar function

$$y = e^{-t^2}$$

$$f(x) = e^{x}$$

$$g(t) = -t^2$$

$$y = f(x) = f(g(t))$$

$$x = g(t)$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

Chain rule: single variable scalar function

$$y = e^{-t^2}$$

$$f(x) = e^x$$

$$g(t) = -t^2$$

$$\frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx}e^x = e^x$$

$$\frac{dx}{dt} = \frac{dg}{dt} = \frac{d}{dt}\left(-t^2\right) = -2t$$

$$y = f(x) = f(g(t))$$

$$x = g(t)$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

Chain rule: single variable scalar function

$$y = e^{-t^2}$$
$$f(x) = e^x$$
$$g(t) = -t^2$$

$$\frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx}e^x = e^x$$
$$\frac{dx}{dt} = \frac{dg}{dt} = \frac{d}{dt}(-t^2) = -2t$$

$$y = f(x) = f(g(t))$$
$$x = g(t)$$
$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt} = e^{-t^2}(-2t)$$

Chain rule: single variable scalar function

$$y = e^{-t^2}$$
$$f(x) = e^x$$
$$g(t) = -t^2$$

$$\frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx}e^x = e^x$$
$$\frac{dx}{dt} = \frac{dg}{dt} = \frac{d}{dt}(-t^2) = -2t$$

$$y = f(x) = f(g(t))$$
$$x = g(t)$$
$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt} = e^{-t^2}(-2t)$$

Common to see both notations

Chain rule: single variable scalar function

$$y = e^{-t^2}$$
$$f(x) = e^x$$
$$g(t) = -t^2$$

$$\frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx}e^x = e^x$$
$$\frac{dx}{dt} = \frac{dg}{dt} = \frac{d}{dt}(-t^2) = -2t$$

$$y = f(x) = f(g(t))$$
$$x = g(t)$$
$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt} = e^{-t^2}(-2t)$$

Chain rule

Chain rule: single variable scalar function

$$y = e^{-t^2}$$

$$f(x) = e^x$$

$$g(t) = -t^2$$

$$\frac{dy}{dx} = \frac{df}{dx} = \frac{d}{dx}e^x = e^x$$

$$\frac{dx}{dt} = \frac{dg}{dt} = \frac{d}{dt}(-t^2) = -2t$$

$$y = f(x) = f(g(t))$$

$$x = g(t)$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

$$\frac{dy}{dt} = \frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt} = e^{-t^2}(-2t)$$

Note: No **x** appears despite the intermediate derivatives

Chain rule: Two variable scalar function

$$z = f(x, y)$$

$$x = g(t) \qquad y = h(t)$$

$$z = f(g(t), h(t))$$

Chain rule: Two variable scalar function

$$z = f(x, y)$$

$$x = g(t) \qquad y = h(t)$$

$$z = f(g(t), h(t))$$

$$\Delta z = \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y$$

A small change in z is the sum of the changes due to x and y changing

Chain rule: Two variable scalar function

$$z = f(x, y)$$

$$x = g(t) \qquad y = h(t)$$

$$z = f(g(t), h(t))$$

$$\Delta z = \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial y}\Delta y$$

$$\frac{dz}{dt} = \frac{df}{dt} = \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt}$$

Need to consider how each variable depends on t

Chain rule: Two variable scalar function

$$z = x^2 + y^2$$

$$x = \sin t$$

$$y = \cos t$$

Chain rule: Two variable scalar function

$$z = x^2 + y^2$$

$$x = \sin t$$

$$y = \cos t$$

$$\begin{aligned}\frac{dz}{dt} = \frac{df}{dt} &= \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} \\ &= (2x)\cos t + (-2y)\sin t \\ &= (2\sin t)\cos t + (-2\cos t)\sin t = 0\end{aligned}$$

Chain rule: Two variable scalar function

$$z = x^2 + y^2$$

$$x = \sin t$$

$$y = \cos t$$

$$\begin{aligned}\frac{dz}{dt} = \frac{df}{dt} &= \frac{\partial f}{\partial x}\frac{dx}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} \\ &= (2x)\cos t + (2y)(-\sin t) \\ &= 2\sin t \cos t + -2\cos t \sin t = 0\end{aligned}$$

$$\frac{df}{dt} = \frac{d}{dt}\left[(\sin t)^2 + (\cos t)^2\right] = \frac{d}{dt}[1] = 0$$

Chain rule: n-variable scalar function chain rule

$$z = f(\mathbf{x})$$

$$\mathbf{x(t)} = [x_1(t), \ldots, x_n(t)]$$

$$\frac{df}{dt} = \sum_i \frac{\partial f}{\partial x_i} \frac{dx_i}{dt} = \nabla f \cdot \frac{d\mathbf{x}}{dt}$$

Chain rule: n-variable scalar function chain rule

$$z = f(\mathbf{x})$$

$$\mathbf{x(t)} = [x_1(t), \ldots, x_n(t)]$$

$$\frac{df}{dt} = \sum_i \frac{\partial f}{\partial x_i} \frac{dx_i}{dt} = \nabla f \cdot \frac{d\mathbf{x}}{dt}$$

Dot-product / matrix multiplication of Jacobians

Chain rule: n-variable scalar function chain rule

$$z = f(\mathbf{x})$$

$$\mathbf{x(t)} = [x_1(t), \ldots, x_n(t)]$$

$$\frac{df}{dt} = \sum_i \frac{\partial f}{\partial x_i} \frac{dx_i}{dt} = \nabla f \cdot \frac{d\mathbf{x}}{dt}$$

$$= \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{d\mathbf{x}}{dt}$$

Dot-product / matrix multiplication of Jacobians

Chain rule: n-variable scalar function chain rule

$$z = f(\mathbf{x}) = (x_1 + x_2 + \cdots + x_n)^2$$

$$\mathbf{x}(t) = (t, t^2, \ldots, t^n)$$

$$z(t) = (t + t^2 + \cdots + t^n)^2$$

Chain rule: n-variable scalar function chain rule

$$z = f(\mathbf{x}) = (x_1 + x_2 + \cdots + x_n)^2$$

$$\mathbf{x}(t) = (t, t^2, \ldots, t^n)$$

$$z(t) = (t + t^2 + \cdots + t^n)^2$$

$$\frac{\partial f}{\partial \mathbf{x}} = 2(t + t^2 + \cdots + t^n)[1, \ldots, 1]$$

$$\frac{\partial \mathbf{x}}{\partial \mathbf{t}} = [1, 2t, \ldots, nt^{n-1}]$$

$$\frac{dz}{dt} = 2(t + t^2 + \cdots + t^n)(1 + 2t + \cdots + nt^{n-1})$$

$$= \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

Chain rule: n-variable scalar function chain rule

$$z = f(\mathbf{x}) = (x_1 + x_2 + \cdots + x_n)^2$$

$$\mathbf{x}(t) = (t, t^2, \ldots, t^n)$$

$$z(t) = (t + t^2 + \cdots + t^n)^2$$

$$\frac{\partial f}{\partial \mathbf{x}} = 2(t + t^2 + \cdots + t^n)[1, \ldots, 1]$$

$$\frac{\partial \mathbf{x}}{\partial \mathbf{t}} = [1, 2t, \ldots, nt^{n-1}]$$

$$\frac{dz}{dt} = 2(t + t^2 + \cdots + t^n)(1 + 2t + \cdots + nt^{n-1})$$

$$= \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

Chain rule: n-variable scalar function chain rule

$$z = f(\mathbf{x}) = (x_1 + x_2 + \cdots + x_n)^2$$

$$\mathbf{x}(t) = (t, t^2, \ldots, t^n)$$

$$z(t) = (t + t^2 + \cdots + t^n)^2$$

$$\frac{\partial f}{\partial \mathbf{x}} = 2(t + t^2 + \cdots + t^n)[1, \ldots, 1]$$

$$\frac{\partial \mathbf{x}}{\partial \mathbf{t}} = [1, 2t, \ldots, nt^{n-1}]$$

$$\frac{dz}{dt} = 2(t + t^2 + \cdots + t^n)(1 + 2t + \cdots + nt^{n-1})$$

$$= \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

Note that we evaluate the first Jacobian at x(t) and the second at t

Chain rule: partial derivatives, two variables

$$z = f(x, y)$$

$$x = g(s, t)$$

$$y = h(s, t)$$

$$z = f(g(s, t), h(s, t))$$

Chain rule: partial derivatives, two variables

$$z = f(x, y)$$
$$x = g(s, t)$$
$$y = h(s, t)$$
$$z = f(g(s, t), h(s, t))$$

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s}$$
$$\frac{\partial f}{\partial t} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t}$$

Same as before but all partials now, and one for each variable

Chain rule: partial derivatives, two variables

$$z = f(x, y) = xy$$

$$x = g(s, t) = s + t$$

$$y = h(s, t) = s - t$$

Chain rule: partial derivatives, two variables

$$z = f(x, y) = xy$$

$$x = g(s, t) = s + t$$

$$y = h(s, t) = s - t$$

$$\begin{aligned} z &= f(g(s, t), h(s, t)) \\ &= (s + t)(s - t) = s^2 - t^2 \end{aligned}$$

Chain rule: partial derivatives, two variables

$$z = f(x, y) = xy$$

$$x = g(s, t) = s + t$$

$$y = h(s, t) = s - t$$

$$z = f(g(s,t), h(s,t))$$
$$= (s+t)(s-t) = s^2 - t^2$$

$$\frac{\partial f}{\partial s} = \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s}$$
$$= y * 1 + x * 1 = y + x = 2s$$

Chain rule: partial derivatives, two variables

$$z = f(x, y) = xy$$

$$x = g(s, t) = s + t$$

$$y = h(s, t) = s - t$$

$$\begin{aligned} z &= f(g(s,t), h(s,t)) \\ &= (s+t)(s-t) = s^2 - t^2 \end{aligned}$$

$$\begin{aligned} \frac{\partial f}{\partial s} &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial s} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial s} \\ &= y * 1 + x * 1 = y + x = 2s \end{aligned}$$

$$\begin{aligned} \frac{\partial f}{\partial t} &= \frac{\partial f}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial f}{\partial y}\frac{\partial y}{\partial t} \\ &= y(1) + x(-1) = y - x = -2t \end{aligned}$$

Chain rule: partial derivatives, n-variables

$$z = f(\mathbf{x})$$

$$\mathbf{x}(\mathbf{t}) = [x_1(\mathbf{t}), \ldots, x_n(\mathbf{t})]$$

$$\mathbf{t} = [t_1, \ldots, t_k]$$

Chain rule: partial derivatives, n-variables

$$z = f(\mathbf{x})$$

$$\mathbf{x}(\mathbf{t}) = [x_1(\mathbf{t}), \ldots, x_n(\mathbf{t})]$$

$$\mathbf{t} = [t_1, \ldots, t_k]$$

$$\frac{\partial f}{\partial t_i} = \sum_j \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial t_i} = \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial t_i}$$

Chain rule: partial derivatives, n-variables

$$z = f(\mathbf{x})$$

$$\mathbf{x}(\mathbf{t}) = [x_1(\mathbf{t}), \ldots, x_n(\mathbf{t})]$$

$$\mathbf{t} = [t_1, \ldots, t_k]$$

$$\frac{\partial f}{\partial t_i} = \sum_j \frac{\partial f}{\partial x_j} \frac{\partial x_j}{\partial t_i} = \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial t_i}$$

All k components fit together into a matrix product of Jacobians

$$\frac{\partial f}{\partial \mathbf{t}} = \frac{\partial f}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

Composition of vector-valued functions

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})]$$

$$\mathbf{x}(\mathbf{t}) = [x_1(\mathbf{t}), \ldots, x_n(\mathbf{t})]$$

$$\mathbf{t} = [t_1, \ldots, t_k]$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{t}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

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

$$x : \mathbb{R}^k \to \mathbb{R}^n$$

$$(f \circ x) = \mathbb{R}^k \to \mathbb{R}^m$$

Again, carefully note the arguments:

$$\frac{\partial f}{\partial \mathbf{t}}(\mathbf{t}) = \frac{\partial f}{\partial \mathbf{x}}(x(\mathbf{t})) \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}(\mathbf{t})$$

Composition of vector-valued functions

$$\mathbf{f}(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})]$$

$$\mathbf{x}(\mathbf{t}) = [x_1(\mathbf{t}), \ldots, x_n(\mathbf{t})]$$

$$\mathbf{t} = [t_1, \ldots, t_k]$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{t}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

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

$$x : \mathbb{R}^k \to \mathbb{R}^n$$

$$(f \circ x) = \mathbb{R}^k \to \mathbb{R}^m$$

Composition of vector-valued functions

$$f : \mathbb{R}^n \to \mathbb{R}^m$$
$$x : \mathbb{R}^k \to \mathbb{R}^n$$
$$(f \circ x) = \mathbb{R}^k \to \mathbb{R}^m$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{t}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}} = \underset{m \times n}{\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}} \underset{n \times k}{\begin{bmatrix} \frac{\partial x_1}{\partial t_1} & \cdots & \frac{\partial x_1}{\partial t_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial t_1} & \cdots & \frac{\partial x_n}{\partial t_k} \end{bmatrix}}$$

Composition of vector-valued functions

$$\frac{\partial \mathbf{f}}{\partial \mathbf{t}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_n} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t_1} & \cdots & \frac{\partial x_1}{\partial t_k} \\ \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial t_1} & \cdots & \frac{\partial x_n}{\partial t_k} \end{bmatrix}$$

$$\frac{df}{dt} = \frac{dy}{dx}\frac{dx}{dt}$$

Single variable scalar function is a very special case where m=n=k=1

Composition of vector-valued functions

Linear functions defined by matrix multiplication

$$f(\mathbf{x}) = A_1\mathbf{x}$$
$$x(\mathbf{t}) = A_2\mathbf{t}$$
$$f(\mathbf{x}(\mathbf{t})) = A_1A_2\mathbf{t}$$

Composition of vector-valued functions

Linear functions defined by matrix multiplication

$$f(\mathbf{x}) = A_1\mathbf{x}$$
$$x(\mathbf{t}) = A_2\mathbf{t}$$
$$f(\mathbf{x}(\mathbf{t})) = A_1A_2\mathbf{t}$$

$$\frac{\partial \mathbf{f}}{\partial \mathbf{t}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}} = A_1A_2$$

Summary: Multivariate Chain rules

- All the chain rules are special cases of the Jacobian product
- Why does the rule work? The derivative is the “best linear approximation”:
  - So the best linear approximation of a composition is the composition of the linear approximations
  - Composition of linear functions is just matrix multiplication
  - So we (matrix) multiply the Jacobians together

$$\frac{\partial \mathbf{f}}{\partial \mathbf{t}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

Summary: Multivariate Chain rules

- All the chain rules are special cases of the Jacobian product
- Why does the rule work? The derivative is the “best linear approximation”:
  - So the best linear approximation of a composition is the composition of the linear approximations
  - Composition of linear functions is just matrix multiplication
  - So we (matrix) multiply the Jacobians together
- Pay special attention to the arguments and notation
  - It’s convenient but easy to make a mistake

$$\frac{\partial \mathbf{f}}{\partial \mathbf{t}} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}$$

$$\frac{\partial f}{\partial \mathbf{t}}(\mathbf{t}) = \frac{\partial f}{\partial \mathbf{x}}(x(\mathbf{t})) \cdot \frac{\partial \mathbf{x}}{\partial \mathbf{t}}(\mathbf{t})$$

Gradient Descent and Neural Networks

Gradient descent on a neural network

A multilayer (feed forward) NN is a composition of such functions

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^1(W^1 \mathbf{x}) \cdots))$$

The exponents are “upper indices” indicating the layer, not powers or iterates of a function or matrix

Gradient descent on a neural network

To fit a neural network, we need to update the network weights based on our data so that the model improves

- Improvement means that the weights change so that our loss or cost function gets smaller
- So we need to perform gradient descent on the vector of all weights using the loss function

Gradient descent on a neural network

To fit a neural network, we need to update the network weights based on our data so that the model improves

- Cost function for input pair

$$C(\mathbf{y}^i, g(\mathbf{x}^i))$$

- Partial derivative for a particular weight

$$\frac{\partial C}{\partial w_{jk}^l}$$

- Update the weight via gradient descent

$$\Delta w_{jk}^l = (w_{jk}^l)' - w_{jk}^l = -\alpha \frac{\partial C}{\partial w_{jk}^l}$$

Activation functions

There are many common activation functions. We've seen the sigmoid:

$$f(x) = \frac{1}{1 + e^{-x}}$$

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

Convenient computationally since we don't have to compute another function to compute the derivative

Example: Single layer, one output

The network is represented by the function

$$g(\mathbf{x}) = f(\mathbf{w} \cdot \mathbf{x} + b) = f(z)$$
$$z = \mathbf{w} \cdot \mathbf{x} + b$$

Let's use squared error for our cost/loss:

$$C(\mathbf{x}, y) = \frac{1}{2}(y - g(\mathbf{x}))^2$$

Partial derivative for one weight:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial g}\frac{\partial g}{\partial z}\frac{\partial z}{\partial w_i} = (y - g(\mathbf{x}))f'(z)x_i$$

Example: Single layer, one output

The network is represented by the function

$$g(\mathbf{x}) = f(\mathbf{w} \cdot \mathbf{x} + b) = f(z)$$
$$z = \mathbf{w} \cdot \mathbf{x} + b$$

Let's use squared error for our cost/loss:

$$C(\mathbf{x}, y) = \frac{1}{2}(y - g(\mathbf{x}))^2$$

Partial derivative for one weight:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial g}\frac{\partial g}{\partial z}\frac{\partial z}{\partial w_i} = (y - g(\mathbf{x}))f'(z)x_i$$

Gradient (all partial derivatives)

$$\nabla_{\mathbf{w}} C = \frac{\partial C}{\partial \mathbf{w}} = \frac{\partial C}{\partial g}\frac{\partial g}{\partial z}\frac{\partial z}{\partial \mathbf{w}} = (y - g(\mathbf{x}))f'(z)\mathbf{x}$$

Example: Single layer, one output

The network is represented by the function

$$g(\mathbf{x}) = f(\mathbf{w} \cdot \mathbf{x} + b) = f(z)$$
$$z = \mathbf{w} \cdot \mathbf{x} + b$$

Substitute in derivative of activation function

$$\begin{aligned} \nabla_{\mathbf{w}} C &= (y - g(\mathbf{x}))f'(z)\mathbf{x} \\ &= (y - f(z))f(z)(1 - f(z))\mathbf{x} \end{aligned}$$

Gradient descent for weights:

$$\mathbf{w}' = \mathbf{w} - \alpha \nabla_{\mathbf{w}} C$$

Example: Two layers, vector output

The network is represented by the function

$$g(\mathbf{x}) = f^2(W^2 f^1(W^1\mathbf{x} + \mathbf{b^1}) + \mathbf{b}^2)$$

Example: Two layers, vector output

The network is represented by the function

$$g(\mathbf{x}) = f^2(W^2 f^1(W^1\mathbf{x} + \mathbf{b^1}) + \mathbf{b}^2)$$

Note several differences:

- Weights are now matrices
- Biases are now vectors
- Layer outputs are now vectors
- Activation functions are applied component-wise
- Exponents are upper indices, not powers
- We reserve lower indices for vector and matrix components

Example: Two layers, vector output

The network is represented by the function

$$g(\mathbf{x}) = f^2(W^2 f^1(W^1\mathbf{x} + \mathbf{b^1}) + \mathbf{b}^2)$$

- Composition is now of vector-valued functions
- Multiple weight matrices so we need to be careful when computing gradients

Example: Two layers, vector output

The network is represented by the function

$$g(\mathbf{x}) = f^2(W^2 f^1(W^1\mathbf{x} + \mathbf{b^1}) + \mathbf{b}^2)$$

- Composition is now of vector-valued functions
- Multiple weight matrices so we need to be careful when computing gradients

Loss function:

$$C(\mathbf{x}, \mathbf{y}) = \frac{1}{2}||\mathbf{y} - g(\mathbf{x})||^2$$

Gradient descent on a neural network

As our networks accrue more layers, there are many more weights to update with gradient descent

It’s very inefficient to naively compute the derivative and update for each weight individually

Instead we want to compute them all efficiently using matrix operations

Backpropagation

Backpropagation

- Backpropagation is the algorithm that allowed neural networks to be trained with practicality

- It's a special case of the backward accumulation of automatic differentiation, which is how JAX and other libraries compute derivatives

Backpropagation

- Backpropagation is the algorithm that allowed neural networks to be trained with practicality
- It's a special case of the backward accumulation of automatic differentiation, which is how JAX and other libraries compute derivatives
- Need to take advantage of how the network is arranged as a sequence of layers to compute derivatives and weight updates rather than as a arbitrary function of a large number of inputs

Backpropagation

- Backpropagation is the algorithm that allowed neural networks to be trained with practicality
- It's a special case of the backward accumulation of automatic differentiation, which is how JAX and other libraries compute derivatives
- Need to take advantage of how the network is arranged as a sequence of layers to compute derivatives and weight updates rather than as a arbitrary function of a large number of inputs
- In other words, we have to use the chain rule, a lot

Backpropagation

- Backpropagation is the algorithm that allowed neural networks to be trained with practicality
- It's a special case of the backward accumulation of automatic differentiation, which is how JAX and other libraries compute derivatives
- Need to take advantage of how the network is arranged as a sequence of layers to compute derivatives and weight updates rather than as a arbitrary function of a large number of inputs
- In other words, we have to use the chain rule, a lot

Backpropagation: Notation

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^2(W^2(f^1(W^1\mathbf{x}))\cdots))$$

$$\begin{aligned}\mathbf{z}^1 &= W^1\mathbf{x} \\ \mathbf{z}^l &= W^l(\text{output from previous layer})\end{aligned}$$

Weighted layer inputs

Backpropagation: Notation

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^2(W^2(f^1(W^1\mathbf{x}))\cdots))$$

$$\mathbf{z}^1 = W^1\mathbf{x}$$
$$\mathbf{z}^l = W^l(\text{output from previous layer})$$

Weighted layer inputs

$$\mathbf{a}^1 = f^1(\mathbf{z}^1) = f^1(W^1\mathbf{x})$$
$$\mathbf{a}^2 = f^2(\mathbf{z}^2) = f^2(W^2\mathbf{a}^1) = f^2(W^2 f^1(W^1\mathbf{x}))$$
$$\mathbf{a}^l = f^l(\mathbf{z}^l)$$

Activated layer outputs

Backpropagation: Forward Pass

$$g(\mathbf{x}) = f^L(W^L f^{L-1}(W^{L-1} \cdots f^2(W^2(f^1(W^1\mathbf{x}))\cdots))$$

$$\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}$$

Compute all the layer inputs and outputs in the forward pass

We'll need these to evaluate the derivatives on the reverse pass

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

Want $\dfrac{\partial C}{\partial \mathbf{W}^l}$ for each layer for gradient descent

We'll compute these iteratively along with a few other necessary derivatives

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

Keep in mind our simpler example:

$$\frac{\partial C}{\partial w_i} = \frac{\partial C}{\partial g}\frac{\partial g}{\partial z}\frac{\partial z}{\partial w_i} = (y - g(\mathbf{x}))f'(z)x_i$$

Our activated outputs play the role of **g** at each layer, so we need to compute the analogous three partials and combine them

Then we progress to the next lower layer continue the derivatives for the next weights

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

$$C(\mathbf{x}, \mathbf{y}) = \frac{1}{2}||\mathbf{y} - g(\mathbf{x})||^2$$

$$\frac{\partial C}{\partial \mathbf{a}^L} = 2(\mathbf{y} - \mathbf{a}^L)$$

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

$$
\frac{\partial C}{\partial \mathbf{z}^L} = \frac{\partial C}{\partial \mathbf{a}^L}\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L}
$$

$$
\frac{\partial \mathbf{a}^L}{\partial \mathbf{z}^L} = \mathrm{Diag}((f^L)'(\mathbf{z}^L))
$$

Diagonal because we apply activation component-wise

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

$$
\frac{\partial C}{\partial \mathbf{a}^{L-1}} = \frac{\partial C}{\partial \mathbf{z}^L}\frac{\partial \mathbf{z}^L}{\partial \mathbf{a}^{L-1}} = \frac{\partial C}{\partial \mathbf{z}^L}W^L
$$

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

Continuing backward, and combining the partials, we have that

$$\frac{\partial C}{\partial \mathbf{W}^l} = \frac{\partial C}{\partial \mathbf{z}^l}\mathbf{a}^{l-1}$$

Backpropagation: Backward Pass

Now we compute the derivatives in reverse and build them up iteratively.

$$
\begin{aligned}
\mathbf{z}^1 &= W^1\mathbf{x} \\
\mathbf{a}^1 &= f^1(\mathbf{z}^1) \\
&\cdots \\
\mathbf{z}^l &= W^l\mathbf{a}^{l-1} \\
\mathbf{a}^l &= f^l(\mathbf{z}^l) \\
&\cdots \\
\mathbf{z}^L &= W^L\mathbf{a}^{L-1} \\
\mathbf{a}^L &= f^L(\mathbf{z}^L) = g(\mathbf{x})
\end{aligned}
$$

Continuing backward, and combining the partials, we have that

$$\frac{\partial C}{\partial \mathbf{W}^l} = \frac{\partial C}{\partial \mathbf{z}^l}\mathbf{a}^{l-1}$$

We then iterate backward through all the layers to get the derivatives for each weight matrix from each layer, using the forward pass computed activated layer outputs

Finally: Gradient Descent!

Once we have the partial derivative of the cost function with respect to each weight in the network, we can use gradient descent to update the weights

$$\Delta w^l_{jk} = (w^l_{jk})' - w^l_{jk} = -\alpha \frac{\partial C}{\partial w^l_{jk}}$$

Good news: various ML libraries do all the heavy lifting for us, so there's no need to compute all these derivatives manually

We'll see an example in the colab

Summary: Backpropagation

- For a given data point
  - Compute forward outputs and activations of each layer
  - Compute the derivatives of the loss function to back propagate the errors (backwards step)
  - Use gradient descent to update the network weights

- Repeat for other data points

Additional resources - Backpropagation

- Detailed example by 3Blue1Brown (video)
- An explicit example (with numbers!) in text by Matt Mazur
- A detailed explanation of the backpropagation algorithm

Further topics

You've got the basic tools now to dig deeper

Constrained optimization

- Lagrange multipliers – finding extrema subject to constraints
  - Also related to autodiff
- Gradient descent on surfaces, manifolds, and other structures
  - Riemannian geometry, differential geometry
  - Natural gradient, Fisher information metric

Improving gradient descent

- Momentum
- Hessian free optimization, conjugate gradients