CIS3333: Mathematics of Machine Learning Fall 2025

Lecture: Multivariate Taylor Series

Date: November 5th, 2025 *Author: Surbhi Goel*

In the previous lectures, we saw that the gradient $\nabla f(x_0)$ provides the best linear approximation to $f : \mathbb{R}^n \to \mathbb{R}$:

$$f(x_0 + h) = f(x_0) + \nabla f(x_0)^\top h + o(\|h\|)$$

This linear view underlies gradient descent. But what if we want a better approximation that captures curvature? The *Taylor series* extends this to polynomial approximations using higher-order derivatives. The second-order version uses the *Hessian* $\nabla^2 f$ and gives the quadratic approximation:

$$f(x_0 + h) = f(x_0) + \nabla f(x_0)^\top h + \frac{1}{2} h^\top \nabla^2 f(x_0) h + o(\|h\|^2)$$

This underlies Newton's method for optimization. Today we'll develop the Hessian and Taylor series, showing how different approximation orders lead to different optimization algorithms.

1 Second Derivatives and the Hessian

The gradient tells us the direction and rate of change. The *Hessian* captures how the gradient itself changes—i.e., the curvature of f.

For $f : \mathbb{R}^2 \to \mathbb{R}$, we can differentiate each component of $\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$ with respect to x and y, giving four second derivatives. These form the Hessian matrix:

$$H = \nabla^2 f(x, y) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix}$$

The Hessian is symmetric: $\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial y \partial x}$ (Clairaut's theorem). Conceptually, the Hessian is the Jacobian of the gradient—it tells us how ∇f changes as we move through space.

1.1 General Case: n Variables

For a scalar function $f : \mathbb{R}^n \to \mathbb{R}$, the Hessian is an $n \times n$ symmetric matrix:

$$\nabla^2 f(x) = H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

Entry (i, j) is: $H_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$

Example. For $f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3$:

- $\nabla f = \begin{bmatrix} 2x_1 x_2 + x_2^3 \\ x_1^2 + 3x_1 x_2^2 \end{bmatrix}$
- $\frac{\partial^2 f}{\partial x_1^2} = 2x_2, \quad \frac{\partial^2 f}{\partial x_2^2} = 6x_1 x_2$
- $\frac{\partial^2 f}{\partial x_1 \partial x_2} = 2x_1 + 3x_2^2 = \frac{\partial^2 f}{\partial x_2 \partial x_1}$
- $H = \begin{bmatrix} 2x_2 & 2x_1 + 3x_2^2 \\ 2x_1 + 3x_2^2 & 6x_1 x_2 \end{bmatrix}$

2 Taylor Series: From Univariate to Multivariate

Now that we understand second derivatives, let's see how they fit into the broader framework of Taylor series—the tool for polynomial approximations of arbitrary order.

2.1 Univariate Taylor Series: Review

Taylor series approximate functions using polynomials by matching derivatives. The degree-n Taylor polynomial at x_0 is:

$$T_n(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k = f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2!}(x - x_0)^2 + \cdots$$

Each term matches the k-th derivative at x_0. For smooth functions, the infinite series

$$f(x) = \sum_{k=0}^{\infty} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

exactly represents f (when it converges). Example: $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$

2.2 Multivariate Taylor Series

How does this extend to functions $f : \mathbb{R}^n \to \mathbb{R}$ of multiple variables? The pattern is similar: we use higher-order derivatives to build polynomial approximations.

First-order (linear) approximation. From Lecture 17, the gradient gives:

$$f(x_0 + h) = f(x_0) + \nabla f(x_0)^\top h + o(\|h\|)$$

where $h = x - x_0$ is the displacement. This is the multivariate version of the tangent line.

Second-order (quadratic) approximation. Including the Hessian:

$$f(x_0 + h) = f(x_0) + \nabla f(x_0)^\top h + \frac{1}{2} h^\top \nabla^2 f(x_0) h + o(\|h\|^2)$$

Let's unpack the second-order term $\frac{1}{2} h^\top H h$ where $H = \nabla^2 f(x_0)$:

- This is a quadratic form: $h^\top H h = \sum_{i=1}^n \sum_{j=1}^n h_i H_{ij} h_j$
- When $n = 1$: $h^\top H h = h^2 H$ reduces to $f''(x_0) h^2$ (univariate case)
- When $n = 2$: $h^\top H h = h_1^2 H_{11} + 2 h_1 h_2 H_{12} + h_2^2 H_{22}$ (quadratic in two variables)
- The factor of $\frac{1}{2}$ comes from the factorial $2! = 2$, matching the univariate formula

General multivariate Taylor series. For a scalar function $f : \mathbb{R}^n \to \mathbb{R}$, the full expansion is:

$$f(x_0 + h) = f(x_0) + \nabla f(x_0)^\top h + \frac{1}{2} h^\top \nabla^2 f(x_0) h + \sum_{k=3}^{\infty} \frac{1}{k!} D^k f(x_0)[h, \ldots, h]$$

where $D^k f(x_0)$ is a k-th order tensor of partial derivatives.

Higher-order derivatives as tensors. Tensors are multidimensional arrays generalizing scalars (order 0), vectors (order 1), and matrices (order 2):

- Order 0: scalar $f(x_0) \in \mathbb{R}$
- Order 1: vector $\nabla f(x_0) \in \mathbb{R}^n$ with entries $\frac{\partial f}{\partial x_i}$
- Order 2: matrix $\nabla^2 f(x_0) \in \mathbb{R}^{n \times n}$ with entries $\frac{\partial^2 f}{\partial x_i \partial x_j}$
- Order k: tensor with n^k entries $\frac{\partial^k f}{\partial x_{i_1} \cdots \partial x_{i_k}}$

The k-th order term in the Taylor series is:

$$\frac{1}{k!} \sum_{i_1, \ldots, i_k = 1}^{n} \frac{\partial^k f}{\partial x_{i_1} \cdots \partial x_{i_k}} h_{i_1} \cdots h_{i_k}$$

a degree-k polynomial in h. In practice, we rarely use beyond second order (the Hessian).

3 Applications of Multivariate Taylor Series

3.1 Optimization Methods

Taylor approximations of different orders lead to different optimization algorithms:

First-order methods (gradient descent). The linear approximation

$$f(x) \approx f(x_0) + \nabla f(x_0)^\top (x - x_0)$$

suggests moving in direction $-\nabla f(x_0)$ to decrease f. This leads to gradient descent and its variants.

Second-order methods (Newton's method). The quadratic approximation

$$f(x) \approx f(x_0) + \nabla f(x_0)^\top (x - x_0) + \frac{1}{2}(x - x_0)^\top \nabla^2 f(x_0)(x - x_0)$$

has a minimum (when $\nabla^2 f(x_0)$ is positive definite). Setting the gradient to zero gives:

$$x = x_0 - [\nabla^2 f(x_0)]^{-1} \nabla f(x_0)$$

This is Newton's method, which accounts for curvature via the Hessian.

3.2 Computing Expectations

For intractable expectations $\mathbb{E}[f(X)] = \int f(x)p(x)\,dx$, approximate f with its Taylor expansion around $\mu = \mathbb{E}[X]$ and integrate term-by-term:

$$\mathbb{E}[f(X)] \approx f(\mu) + \frac{1}{2}\text{tr}\left(\nabla^2 f(\mu)\,\text{Cov}(X)\right)$$

The first-order term vanishes since $\mathbb{E}[X - \mu] = 0$. This is used in the Extended Kalman Filter and Laplace approximation.

4 Why First-Order Methods for Neural Networks?

Given that Newton's method converges faster than gradient descent, why don't we use it for training neural networks?

Computational complexity. For a model with n parameters:

Operation	**Memory**	**Time per iteration**
Gradient (backprop)	$O(n)$	$O(n)$
Hessian	$O(n^2)$	$O(n^2)$
Hessian inverse	$O(n^2)$	$O(n^3)$

The scale problem. For $n = 10^6$ parameters, the Hessian requires $(10^6)^2 = 10^{12}$ entries (4 terabytes!) and $O(n^3)$ time to invert. Modern networks have millions to billions of parameters, making second-order methods infeasible.

Practical alternatives. Instead, we use:

- **First-order methods:** SGD, Adam (only need $O(n)$ gradients via backprop)
- **Quasi-Newton:** L-BFGS approximates $[\nabla^2 f]^{-1}$ with $O(n)$ memory
- **Adaptive methods:** Adam approximates per-parameter curvature without full Hessian

Second-order methods remain valuable for small-parameter problems or when using Hessian approximations. But backpropagation's efficient gradient computation is why first-order methods dominate deep learning.