Chapter 3: Calculus and Optimization Essentials for Machine Learning

Youssef SALMAN

Applied Mathematics
Lebanese University — Faculty of Sciences (Section V)

M1: Applied Mathematics & Statistics for Computational Sciences

September 15, 2025

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

Why this chapter

- We turn models into **decisions**: choose parameters that minimize a loss (possibly under constraints).
- Calculus provides the **tools** (gradients, Hessians, Taylor) used by optimization.
- Optimization provides the **updates**: gradient descent (full, stochastic, mini-batch), Newton, projections.
- Focus today: **intuition + Python**. No formal proofs; everything connects to runnable code.

Roadmap

- Single-variable recap: limits, continuity, derivative (why ML cares).
- Multivariate essentials: gradient, Jacobian, Hessian, directional derivative, Taylor.
- Smoothness & convexity: what they guarantee and why they matter for steps.
- Optimization basics: objective, variants of gradient descent, Newton, constraints via projection.
- Liveable code snippets to **see** the behavior.

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

Limits, continuity, derivative (quick recap)

- Limit $\lim_{x \to a} f(x)$: value approached by f when $x \to a$.
- Continuous at a if $\lim_{x \to a} f(x) = f(a)$.
- Derivative $f'(a) = \lim_{h \to 0} \dfrac{f(a+h) - f(a)}{h}$ (instantaneous rate of change).
- **Why ML cares:** derivatives quantify marginal change of the loss when we tweak a weight.

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

Gradient, Jacobian, Hessian

- Gradient $\nabla f(x) \in \mathbb{R}^p$: direction of steepest *increase*.
- Jacobian $J_g(x) \in \mathbb{R}^{m \times p}$: partials of vector-valued g.
- Hessian $H_f(x) \in \mathbb{R}^{p \times p}$: second-order partials (curvature).
- Directional derivative $D_u f(x) = \nabla f(x)^\top u$ (rate of change along u).

Chain rule (vector view) & Taylor intuition

- Chain rule: for $f = g \circ h$, $\nabla f(x) = [J_h(x)]^\top \nabla g(h(x))$.
- First-order Taylor: $f(x+s) \approx f(x) + \nabla f(x)^\top s$.
- Second-order Taylor: $f(x+s) \approx f(x) + \nabla f(x)^\top s + \frac{1}{2} s^\top H_f(x) s$.
- **Takeaway:** gradient gives direction; Hessian scales the step by curvature.

Critical points, smoothness, convexity

- Critical points: $\nabla f(x^*) = 0$.
  - $H_f(x^*) \succ 0$: local min; $H_f(x^*) \prec 0$: local max; indefinite: saddle.
- L-smooth (Lipschitz gradient): controls how fast gradient changes $\Rightarrow$ safe steps.
- Convexity (calculus view): $H_f(x) \succeq 0$; strong convexity: $H_f(x) \succeq \mu I$.
- **Implication:** convex problems have no spurious local minima (great for training).

Useful ML derivatives (you will reuse them)

- $\nabla_x (a^\top x) = a, \quad \nabla_x \frac{1}{2}\|x\|^2 = x.$
- Least squares: $J(w) = \frac{1}{2n}\|Xw - y\|^2, \ \nabla J(w) = \frac{1}{n} X^\top (Xw - y).$
- Logistic (binary): $J(w) = \frac{1}{n}\sum[-y \log p - (1-y)\log(1-p)]$, $p = \sigma(Xw) \Rightarrow \nabla J(w) = \frac{1}{n} X^\top (p - y)$, Hessian $X^\top \mathrm{diag}(p(1-p))X/n$.

Worked example we will optimize later

- $f(x_1, x_2) = x_1^2 + 2x_2^2 + e^{-x_1 - x_2}$.
- $\nabla f = \left[2x_1 - e^{-x_1-x_2},\ 4x_2 - e^{-x_1-x_2}\right]^\top$.
- $H_f = \begin{bmatrix} 2 + e^{-x_1-x_2} & e^{-x_1-x_2} \\ e^{-x_1-x_2} & 4 + e^{-x_1-x_2} \end{bmatrix} \succ 0$ (unique min).

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

What is optimization in ML?

- Choose parameters x to minimize an objective $f(x)$ (empirical risk + regularization).
- May have constraints $x \in C$ (box norms, budgets, domain rules).
- Algorithms = **update rules** that move x to reduce f (or satisfy constraints).

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

Same update, different gradients

- Canonical update: $x_{k+1} = x_k - \eta\, g_k$.
- **Only difference** is how we compute g_k:
  - **Full-batch:** $g_k = \nabla f(x_k)$ (exact gradient).
  - **SGD:** one sample's gradient (noisy but cheap).
  - **Mini-batch:** average over m samples (trade variance/speed).

Full-batch gradient descent on $y = x^2$

```
import numpy as np
import matplotlib.pyplot as plt

def y_function(x): return x**2
def y_derivative(x): return 2*x

x = np.arange(-100, 100, 0.1)
y = y_function(x)

current_pos = (50, y_function(50))
eta = 0.01

plt.ion()
for _ in range(1000):
    new_x = current_pos[0] - eta * y_derivative(current_pos[0])
    new_y = y_function(new_x)
    current_pos = (new_x, new_y)
    plt.plot(x, y); plt.scatter(current_pos[0], current_pos[1], color="red")
    plt.pause(0.001); plt.clf()
plt.ioff(); plt.show()
```

Stochastic GD on the same $y = x^2$

```
import numpy as np
import matplotlib.pyplot as plt

def f(x): return x**2

rng = np.random.default_rng(0)
xx = np.arange(-100, 100, 0.1); yy = f(xx)

x, eta, steps = 80.0, 0.01, 1000
plt.ion()
for t in range(steps):
    z = rng.normal(0.0, 1.0) # one sample Z ~ N(0,1)
    g = 2*(x - z) # unbiased noisy gradient (E[g|x]=2x)
    x = x - eta*g
    plt.plot(xx, yy); plt.scatter(x, f(x), color="red")
    plt.title(f"SGD on x^2 | iter={t} | x={x:.2f}")
    plt.pause(0.001); plt.clf()
plt.ioff(); plt.show()
```

Mini-batch GD on the same $y = x^2$ (batch size m)

```
import numpy as np
import matplotlib.pyplot as plt

def f(x): return x**2
xx = np.arange(-100, 100, 0.1); yy = f(xx)

x, eta, steps, m = 80.0, 0.01, 600, 32
rng = np.random.default_rng(0)

plt.ion()
for t in range(steps):
    z_batch = rng.normal(0.0, 1.0, size=m)
    g = 2.0*(x - z_batch.mean()) # average of m samples
    x = x - eta*g
    plt.plot(xx, yy); plt.scatter(x, f(x), color="red")
    plt.xlim(-100,100); plt.ylim(0,10000)
    plt.xlabel("x"); plt.ylabel("f(x)=x^2")
    plt.title(f"Mini-batch | m={m} | iter={t} | x={x:.2f}")
    plt.pause(0.001); plt.clf()
plt.ioff(); plt.show()
```

Comparing the three (same objective)

- **Full-batch:** `g = 2*x` (steady, accurate).
- **SGD:** `g = 2*(x - z)` with `z ~ N(0,1)` (noisy, very cheap).
- **Mini-batch:** `g = 2*(x - mean(z_batch))` (variance $\downarrow$ as $m \uparrow$).
- All are **unbiased** for $2x$; batch size controls noise/smoothness vs. speed.

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

Newton in one slide (what/why)

- Build a local **quadratic** model using gradient and Hessian.
- Step solves $H_k\, s_k = g_k$; update $x_{k+1} = x_k - s_k$.
- **Why care:** near a well-conditioned minimum, convergence can be very fast.
- Practical note: solve linear system; don't invert H_k. Damping if needed.

Python: one Newton step for logistic regression

```
import numpy as np

def sigmoid(z): return 1.0/(1.0 + np.exp(-z))

def logreg_grad(X, y, w, lam=0.0):
    p = sigmoid(X @ w)
    return (X.T @ (p - y))/len(y) + lam*w

def logreg_hess(X, y, w, lam=0.0):
    p = sigmoid(X @ w)
    W = p*(1-p) # length-n
    H = (X.T * W) @ X / len(y) + lam*np.eye(X.shape[1])
    return H

def newton_update(X, y, w, lam=0.0):
    g = logreg_grad(X, y, w, lam)
    H = logreg_hess(X, y, w, lam)
    delta = np.linalg.solve(H, g) # solve H*delta = g
    return w - delta
```

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

Convex sets & functions (why they are friendly)

- Convex set: any segment between two points stays inside.
- Convex function: lies below its chords; no spurious local minima.
- Strong convexity: unique minimizer, helpful growth properties.
- Subgradients handle non-smooth convex losses (e.g., ℓ_1).

Constrained view & projection idea

- Problem: $\min_{x \in C} f(x)$ with closed convex C.
- Projected step: $x_{k+1} = \text{proj}_C\big(x_k - \eta \nabla f(x_k)\big)$.
- For box constraints, projection = coordinate-wise clipping.

Python: projected gradient on a convex quadratic

```
import numpy as np
import matplotlib.pyplot as plt

Q = np.array([[3.0, 1.0],
              [1.0, 2.0]])
c = np.array([-2.0, 1.0])

def f(x): return 0.5 * x.T @ Q @ x + c.T @ x
def grad(x): return Q @ x + c

a = np.array([-2.0, -2.0]); b = np.array([2.0, 2.0])
def proj_box(z): return np.minimum(np.maximum(z, a), b)

L = np.max(np.linalg.eigvals(Q).real)
eta = 1.0 / L

x = np.array([1.8, -1.5]); xs = [x.copy()]
for _ in range(100):
    x = proj_box(x - eta*grad(x)); xs.append(x.copy())

xs = np.array(xs)
gx = np.linspace(-2.2, 2.2, 200); gy = np.linspace(-2.2, 2.2, 200)
X, Y = np.meshgrid(gx, gy)
Z = 0.5*(Q[0,0]*X**2 + 2*Q[0,1]*X*Y + Q[1,1]*Y**2) + c[0]*X + c[1]*Y
```

Outline

1. Motivation & Roadmap
2. Single-variable recap
3. Multivariate calculus essentials
4. Optimization basics
5. Gradient descent family
6. Newton's method
7. Convex optimization
8. Takeaways

What to remember

- **Calculus $\Rightarrow$ Optimization:** gradient = direction, Hessian = curvature.
- **GD family:** same update, different gradient estimators (exact vs. noisy vs. averaged).
- **Newton:** curvature-aware steps, solve linear system; damp if unstable.
- **Constraints:** projection is just clipping for boxes; easy to implement.
- All concepts showed up in **runnable Python**, ready for labs/assignments.

Next steps

- Practice: tweak learning rates, batch sizes; observe trajectories.
- Extend: add momentum, line search, and L1 penalties (proximal step).
- Apply: plug these updates into real ML tasks (linear/logistic regression, shallow nets).