

Machine Learning

Textbook: Machine Learning Refined, by Watt, Borhani, and Katsaggelos, 2nd edition:
MLRefined.com

Previous Lecture

- ◆ Second-order Taylor approximation adds a curvature term to the local linear model.
- ◆ The Hessian contains all second partial derivatives of the cost function.
- ◆ $\frac{1}{2} \Delta \mathbf{w}^T H \Delta \mathbf{w}$ is the quadratic curvature correction.
- ◆ $\mathbf{u}^T H \mathbf{u}$ measures curvature in direction $\mathbf{u}$.
- ◆ Hessian eigenvectors give principal curvature directions; eigenvalues give the corresponding curvatures.
- ◆ At a stationary point, positive-definite Hessian $\Rightarrow$ local minimum; indefinite Hessian $\Rightarrow$ saddle.



§4.3,4.4 - Newton's Method in Several Variables

Last time:

gradient = slope,
Hessian = curvature.

Together they give a local quadratic model.

Today:

minimize that quadratic model directly.

Start in One Variable

Near the current point w :

$$g(w + \Delta w) \approx g(w) + g'(w)\Delta w + \frac{1}{2}g''(w)\Delta w^2.$$

Call this quadratic model

$$q(\Delta w) = g(w) + g'(w)\Delta w + \frac{1}{2}g''(w)\Delta w^2.$$

Instead of choosing a small step manually:

minimize q .

Minimize the Quadratic Model

Differentiate with respect to the step:

$$q'(\Delta w) = g'(w) + g''(w)\Delta w.$$

Set equal to zero:

$$g'(w) + g''(w)\Delta w = 0.$$

Thus

$$\Delta w = -\frac{g'(w)}{g''(w)}.$$

The new point is

$$w_{\text{new}} = w - \frac{g'(w)}{g''(w)}.$$

This is **Newton’s method** for optimization.

Compare With Gradient Descent

Gradient descent:

$$w_{\text{new}} = w - \alpha g'(w).$$

Newton:

$$w_{\text{new}} = w - \frac{g'(w)}{g''(w)}.$$

Gradient descent:

chooses a fixed or externally selected scale α .

Newton:

uses curvature to scale the gradient automatically.

Large curvature:

smaller Newton step.

Small curvature:

larger Newton step.

A Quadratic Example

Let

$$g(w) = 5(w - 3)^2.$$

Then

$$\begin{aligned} g'(w) &= 10(w - 3), \\ g''(w) &= 10. \end{aligned}$$

Newton step:

$$\Delta w = -\frac{10(w-3)}{10} = -(w - 3).$$

Therefore

$$w_{\text{new}} = w - (w - 3) = 3.$$

From **any** starting point:

one Newton step reaches the minimum.

Why One Step for a Quadratic?

For a quadratic function,

the second-order Taylor model is exact.

Newton minimizes that model.

Therefore:
quadratic function + invertible positive curvature
 $\Rightarrow$
Newton reaches the minimizer in one step.

For a nonquadratic function:
the quadratic model is only local.

So we rebuild it after every step.

A Nonlinear Example

Let
$$g(w) = e^w - w.$$

Then
$$g'^w - 1,$$
$$g''^w.$$

The minimum occurs where
$$e^w - 1 = 0,$$

so
$$w^* = 0.$$

Newton gives
$$w_{k+1} = w_k - \frac{e^{w_k}-1}{e^{w_k}}.$$

Newton Iterations by Hand

Start with
$$w_0 = 1.$$

First step:
$$w_1 = 1 - \frac{e-1}{e}$$
$$= \frac{1}{e} \approx 0.36788.$$

Continuing:
$$w_2 \approx 0.06008,$$
$$w_3 \approx 0.001769,$$
$$w_4 \approx 0.00000156.$$

Very rapid convergence near the minimum.

For a well-behaved problem near a nondegenerate minimum:
Newton often converges **quadratically**.

Roughly:
number of correct digits can double each iteration.

Another View: Solve $g'(w) = 0$

A minimum usually satisfies

$$g'(w) = 0.$$

Apply ordinary Newton root-finding to

$$f(w) = g'(w).$$

Newton root-finding says

$$w_{\text{new}} = w - \frac{f(w)}{f'(w)}.$$

But

$$f(w) = g'(w),$$

$$f'(w) = g''(w).$$

Therefore

$$w_{\text{new}} = w - \frac{g'(w)}{g''(w)}.$$

Same method.

Now Several Parameters

From Lesson 1.7:

$$g(\mathbf{w} + \Delta \mathbf{w}) \approx g(\mathbf{w}) + \nabla g(\mathbf{w})^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T H(\mathbf{w}) \Delta \mathbf{w}.$$

Define the quadratic model

$$q(\Delta \mathbf{w}) = g(\mathbf{w}) + \nabla g^T \Delta \mathbf{w} + \frac{1}{2} \Delta \mathbf{w}^T H \Delta \mathbf{w}.$$

Now minimize with respect to the vector

$$\Delta \mathbf{w}.$$

Differentiate the Quadratic Model

The gradient of q with respect to $\Delta \mathbf{w}$ is

$$\nabla_{\Delta \mathbf{w}} q = \nabla g + H \Delta \mathbf{w}.$$

Set equal to zero:

$$\nabla g + H \Delta \mathbf{w} = \mathbf{0}.$$

Therefore the Newton step satisfies

$$H(\mathbf{w}) \Delta \mathbf{w} = -\nabla g(\mathbf{w}).$$

If H is invertible:

$$\Delta \mathbf{w} = -H(\mathbf{w})^{-1} \nabla g(\mathbf{w}).$$

Thus

$$\mathbf{w}_{k+1} = \mathbf{w}_k - H(\mathbf{w}_k)^{-1} \nabla g(\mathbf{w}_k).$$

But Do Not Actually Compute the Inverse

The formula contains

$$H^{-1}\nabla g.$$

Numerically, we usually do **not** form H^{-1} .

Instead solve

$$H\mathbf{s} = -\nabla g$$

for the step $\mathbf{s}$.

Then

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \mathbf{s}.$$

Solving the linear system is generally more stable and efficient.

Return to Our Familiar Quadratic

Use

$$g(w_1, w_2) = (w_1 - 2)^2 + 4(w_2 + 1)^2.$$

Gradient:

$$\nabla g = \begin{bmatrix} 2(w_1 - 2) \\ 8(w_2 + 1) \end{bmatrix}.$$

Hessian:

$$H = \begin{bmatrix} 2 & 0 \\ 0 & 8 \end{bmatrix}.$$

Start at

$$\mathbf{w}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Then

$$\nabla g(\mathbf{w}_0) = \begin{bmatrix} -4 \\ 8 \end{bmatrix}.$$

Solve for the Newton Step

Solve

$$\begin{bmatrix} 2 & 0 \\ 0 & 8 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = - \begin{bmatrix} -4 \\ 8 \end{bmatrix}.$$

So

$$\begin{aligned} 2s_1 &= 4, \\ 8s_2 &= -8. \end{aligned}$$

Thus

$$\mathbf{s} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}.$$

Therefore

$$\begin{aligned} \mathbf{w}_1 &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 2 \\ -1 \end{bmatrix} \\ &= \begin{bmatrix} 2 \\ -1 \end{bmatrix}. \end{aligned}$$

One step to the exact minimizer.

Why Newton Handles Unequal Curvature Better

Gradient descent step:

$$-\alpha \begin{bmatrix} 2(w_1 - 2) \\ 8(w_2 + 1) \end{bmatrix}.$$

The steep w_2 direction has a much larger gradient.

Newton multiplies by

$$H^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/8 \end{bmatrix}.$$

So it rescales the directions:

$$\begin{aligned} -H^{-1}\nabla g &= -\begin{bmatrix} 1/2 & 0 \\ 0 & 1/8 \end{bmatrix} \begin{bmatrix} 2(w_1 - 2) \\ 8(w_2 + 1) \end{bmatrix} \\ &= \begin{bmatrix} 2 - w_1 \\ -1 - w_2 \end{bmatrix}. \end{aligned}$$

Curvature automatically corrects the scaling.

Newton in the Hessian Eigenvector Basis

Suppose

$$H\mathbf{v}_i = \lambda_i \mathbf{v}_i.$$

Write the gradient as

$$\nabla g = \sum_i c_i \mathbf{v}_i.$$

Then

$$H^{-1}\nabla g = \sum_i \frac{c_i}{\lambda_i} \mathbf{v}_i.$$

So Newton divides each gradient component by its curvature.

Steep direction:

large $\lambda_i \Rightarrow$ smaller step component.

Shallow direction:
small $\lambda_i \Rightarrow$ larger step component.

This directly addresses poor conditioning.

A Quadratic With Cross Terms

Let
 $g(w_1, w_2) = w_1^2 + 2w_1w_2 + 2w_2^2 - 4w_1 - 6w_2.$

Gradient:
$$\nabla g = \begin{bmatrix} 2w_1 + 2w_2 - 4 \\ 2w_1 + 4w_2 - 6 \end{bmatrix}.$$

Hessian:
$$H = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}.$$

Start at
 $(0, 0).$

Then
$$\nabla g(0, 0) = \begin{bmatrix} -4 \\ -6 \end{bmatrix}.$$

Newton Step With Coupled Parameters

Solve
$$\begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 6 \end{bmatrix}.$$

Equations:
 $2s_1 + 2s_2 = 4,$
 $2s_1 + 4s_2 = 6.$

Subtract:
 $2s_2 = 2,$
 $s_2 = 1.$

Then
 $s_1 = 1.$

So
$$\mathbf{w}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Because g is quadratic:

this is the exact minimizer.

Newton as a System of Equations

A stationary point satisfies

$$\nabla g(\mathbf{w}) = \mathbf{0}.$$

Apply Newton's method to this vector equation.

The Jacobian of

$$\nabla g$$

is

$$H.$$

Newton for a vector equation says

$$H(\mathbf{w}_k)\Delta\mathbf{w} = -\nabla g(\mathbf{w}_k).$$

Same Newton step.

Again:

optimization Newton

=

root-finding Newton applied to the gradient.

When Newton Works Very Well

Near a strict local minimum where

$$H(\mathbf{w}^*)$$

is positive definite and nonsingular,

and the function is sufficiently smooth:

Newton can converge extremely rapidly.

Advantages:

no fixed learning rate required in the basic formula;

curvature corrects scaling;

quadratic problems are solved in one step.

But there are costs.

Problem 1: The Hessian Can Be Singular

Suppose

$$g(w) = w^4.$$

Then

$$g'^3,$$

$$g''^2.$$

At the minimum $w = 0$:
 $g''(0) = 0$.

The Newton formula would divide by zero.

In several variables:
singular Hessian $\Rightarrow$ linear system may have no unique Newton step.

Problem 2: The Hessian May Not Describe a Bowl

Consider
 $g(w) = -w^2$.

Then
 $g'(w) = -2w$,
 $g''(w) = -2$.

Starting at $w = 1$:
 $w_{\text{new}} = 1 - \frac{-2}{-2} = 0$.

Newton moves directly to
 $w = 0$.

But $w = 0$ is a **maximum**.

Newton solves for a stationary point.

It does not automatically know we want a minimum.

Problem 3: Far From the Minimum

The quadratic model is local.

Far from a minimum:
it may be a poor approximation.

A full Newton step can be too aggressive.

A common modification:

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha_k \mathbf{s}_k,$$

where
 $H(\mathbf{w}_k) \mathbf{s}_k = -\nabla g(\mathbf{w}_k)$

and
 $0 < \alpha_k \leq 1$.

This is a **damped Newton step**.

A line search can choose α_k .

Problem 4: The Hessian Is Expensive

With d parameters:
gradient has d entries.

Hessian has
 d^2 entries.

A dense linear solve typically costs on the order of
 d^3
operations.

For a model with millions of parameters:
impossible to form and solve with the full Hessian directly.

This is why large-scale machine learning often favors
first-order or approximate second-order methods.

Newton’s Method in Python

For our familiar quadratic:

```
def g(w):  
    return (w[0] - 2)**2 + 4*(w[1] + 1)**2  
  
def grad_g(w):  
    return np.array([2*(w[0] - 2),  
                    8*(w[1] + 1)])  
  
def hess_g(w):  
    return np.array([[2.0, 0.0],  
                    [0.0, 8.0]])
```

Start at
`w = np.array([0.0, 0.0])`

Solve — Do Not Invert

Compute the Newton step:

```
H = hess_g(w)  
grad = grad_g(w)  
  
step = np.linalg.solve(H, -grad)
```

Then update:
`w = w + step`
`w`

Output:
`[ 2. -1.]`

New Python idea:
`np`

solves
 $A\mathbf{x} = \mathbf{b}$.

Prefer this to explicitly computing
np.

A Newton Loop

For a nonquadratic function:

```
w = np.array([1.0])

for k in range(6):
    grad = grad_g(w)
    H = hess_g(w)

    step = np.linalg.solve(H, -grad)
    w = w + step
```

Same structure as gradient descent:

- evaluate local information;
- compute a step;
- update;
- repeat.

Difference:
Newton uses both gradient and Hessian.

Gradient Descent Versus Newton

	Gradient Descent	Newton
Information	∇g	∇g and H
Step	$-\alpha \nabla g$	solve $H\mathbf{s} = -\nabla g$
Curvature	ignored	used directly
Quadratic cost	many steps possible	one step
Per iteration	relatively cheap	relatively expensive
Large d	often practical	full method often impractical

Neither method dominates in every setting.

The computational scale matters.

Why Learn Newton if We Rarely Use the Full Hessian?

Newton’s method gives the ideal second-order picture.

It explains why curvature matters.

It motivates methods that try to obtain some of Newton’s benefits

without storing or solving with the full Hessian.

Examples later in optimization:

quasi-Newton methods;

approximate curvature;

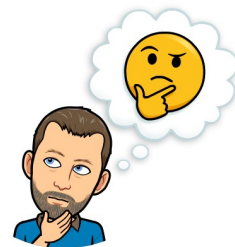
Hessian-vector products.

For now:

Newton completes our basic optimization toolkit.

What did we learn?

- ◆ Newton's method chooses the step that minimizes the local quadratic model.
- ◆ In one variable, $\Delta w = -g'(w)/g''(w)$.
- ◆ In several variables, the Newton step solves $H\Delta \mathbf{w} = -\nabla g$.
- ◆ Newton rescales gradient components according to curvature and solves a positive-definite quadratic in one step.
- ◆ Newton's method can fail when the Hessian is singular, indefinite, or when the quadratic model is poor far from the solution.
- ◆ Numerically, use a linear solve rather than explicitly forming H^{-1} .



Next

- We now have zero-order, first-order, and second-order optimization ideas.
- Next we begin using them to fit an actual machine-learning model.
- **1.9 - Linear Regression as Machine Learning**