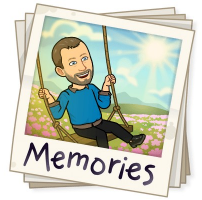


Machine Learning

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

Previous Lecture

- ◆ The gradient collects all first partial derivatives of the cost function.
- ◆ $g(\mathbf{w} + \Delta\mathbf{w}) \approx g(\mathbf{w}) + \nabla g(\mathbf{w})^T \Delta\mathbf{w}$.
- ◆ $\nabla g \cdot \mathbf{u}$ gives the directional derivative in direction $\mathbf{u}$.
- ◆ The gradient points in the direction of steepest local increase; $-\nabla g$ points downhill.
- ◆ At a smooth interior minimum, $\nabla g = \mathbf{0}$, but a zero gradient can also occur at a saddle point.



§3.5 - Gradient Descent

Last time: the gradient points in the direction of steepest local increase.

So $-\nabla g$ points downhill.

Today: turn that direction into an algorithm.

From Direction to a Step

At the current parameter vector $\mathbf{w}$,
steepest descent direction = $-\nabla g(\mathbf{w})$.

Take a step of size $\alpha > 0$:

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

Therefore the new parameter vector is

$$\mathbf{w}_{\text{new}} = \mathbf{w} - \alpha \nabla g(\mathbf{w}).$$

α is called the **step size** or **learning rate**.

Why Does This Move Downhill?

From the local linear approximation:

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

Use

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

Then

$$g(\mathbf{w} - \alpha \nabla g) \approx g(\mathbf{w}) - \alpha \nabla g^T \nabla g.$$

Since

$$\nabla g^T \nabla g = \|\nabla g\|^2,$$

we get

$$g(\mathbf{w} - \alpha \nabla g) \approx g(\mathbf{w}) - \alpha \|\nabla g(\mathbf{w})\|^2.$$

If $\nabla g \neq \mathbf{0}$ and α is sufficiently small:
cost decreases.

Gradient Descent

Repeat the downhill step.

Starting with $\mathbf{w}_0$:

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

Each iteration:

1. evaluate the gradient;
2. move opposite the gradient;
3. repeat.

This is **gradient descent**.

Start in One Variable

Let

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

Then

$$g'(w) = 2(w - 3).$$

Gradient descent becomes

$$w_{k+1} = w_k - \alpha 2(w_k - 3).$$

Choose

$$w_0 = 0, \quad \alpha = 0.2.$$

First step:

$$\begin{aligned} g'(0) &= -6, \\ w_1 &= 0 - 0.2(-6) = 1.2. \end{aligned}$$

The cost drops from

$$\begin{aligned} g(0) &= 9 \\ \text{to} \\ g(1.2) &= 3.24. \end{aligned}$$

Keep Going

Second step:

$$\begin{aligned} g'(1.2) &= 2(1.2 - 3) = -3.6, \\ w_2 &= 1.2 - 0.2(-3.6) = 1.92. \end{aligned}$$

Third step:

$$\begin{aligned} g'(1.92) &= -2.16, \\ w_3 &= 1.92 - 0.2(-2.16) = 2.352. \end{aligned}$$

Fourth step:
 $w_4 = 2.6112$.

The iterates move toward
 $w^* = 3$.

k	0	1	2	3	4
w_k	0	1.2	1.92	2.352	2.6112
$g(w_k)$	9	3.24	1.1664	0.4199	0.1512

Return to Two Parameters

Use our familiar cost function:
 $g(w_1, w_2) = (w_1 - 2)^2 + 4(w_2 + 1)^2$.

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

Start at
$$\mathbf{w}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix},$$

with $\alpha = 0.1$.

At $\mathbf{w}_0$:
$$\nabla g(0, 0) = \begin{bmatrix} -4 \\ 8 \end{bmatrix}.$$

So
$$\mathbf{w}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 0.1 \begin{bmatrix} -4 \\ 8 \end{bmatrix} = \begin{bmatrix} 0.4 \\ -0.8 \end{bmatrix}.$$

Did the Cost Decrease?

Initial cost:
 $g(0, 0) = 8$.

After one gradient step:
$$\begin{aligned} g(0.4, -0.8) &= (-1.6)^2 + 4(0.2)^2 \\ &= 2.56 + 0.16 = 2.72. \end{aligned}$$

One step:
 $8 \rightarrow 2.72$.

At the new point:

$$\nabla g(0.4, -0.8) = \begin{bmatrix} -3.2 \\ 1.6 \end{bmatrix}.$$

Next iterate:

$$\mathbf{w}_2 = \begin{bmatrix} 0.4 \\ -0.8 \end{bmatrix} - 0.1 \begin{bmatrix} -3.2 \\ 1.6 \end{bmatrix} = \begin{bmatrix} 0.72 \\ -0.96 \end{bmatrix}.$$

The Gradient Changes as We Move

Gradient descent does **not** choose one direction and follow it forever.

At every new point:
recompute ∇g .

The direction changes with the local shape of the cost surface.

For our example:

k	$\mathbf{w}_k$	$g(\mathbf{w}_k)$
0	(0, 0)	8
1	(0.4, -0.8)	2.72
2	(0.72, -0.96)	1.6448
3	(0.976, -0.992)	1.0488

The path bends toward
(2, -1).

Why Not Normalize the Gradient?

Last time we found the unit steepest-descent direction:

$$-\frac{\nabla g}{\|\nabla g\|}.$$

Standard gradient descent usually uses

$-\nabla g$
instead.

Why?

The magnitude $\|\nabla g\|$ contains useful information.

Steep region $\Rightarrow$ larger gradient $\Rightarrow$ larger move.

Near a stationary point $\Rightarrow$ smaller gradient $\Rightarrow$ smaller move.

The learning rate α controls the overall scale.

The Learning Rate Matters

Gradient descent uses

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

If α is very small:
progress may be slow.

If α is too large:
we may jump across the valley.

Possibly repeatedly.

Possibly with increasing cost.

How should we choose α ?

Next lesson.

Gradient Descent in Python

Represent the parameter vector as one NumPy array:

```
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)])
```

Start at (0,0):

```
w = np.array([0.0, 0.0])
alpha = 0.1
```

One gradient-descent step:

```
w = w - alpha*grad_g(w)
w
```

Output:

```
[ 0.4 -0.8]
```

Exactly what we found by hand.

Now Repeat the Step

A **for loop** lets us repeat the update:

```
w = np.array([0.0, 0.0])
alpha = 0.1

for k in range(10):
    print(k, w, g(w))
    w = w - alpha*grad_g(w)
```

New Python idea:

range produces 0,1,...,9.

The indented lines repeat once for each value of k .

This is our first actual learning loop.

Save the Iterates

Sometimes we want the whole path:

```
w = np.array([0.0, 0.0])
history = [w.copy()]

for k in range(20):
    w = w - alpha*grad_g(w)
    history.append(w.copy())
```

New Python idea:

- a **list** can store successive parameter vectors.

w stores the current values of w .

Later we can plot the path through parameter space.

When Do We Stop?

Possible stopping rules:

- $\|\nabla g(\mathbf{w}_k)\|$ is small;
- the change in cost is small;
- the change in parameters is small;
- or a maximum number of iterations is reached.

For now:

- use a fixed number of iterations.

What Gradient Descent Does, and Does Not, Promise

Gradient descent is a **local** method.

It uses the gradient at the current point.

For a simple bowl-shaped cost:

- we expect movement toward the minimum.

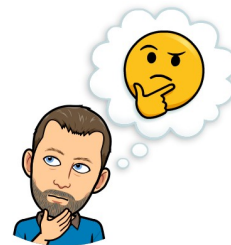
For a complicated landscape:

- different starting points may behave differently.

A stationary point need not be the global minimum.

What did we learn?

- ◆ Gradient descent repeatedly moves opposite the gradient.
- ◆ $\mathbf{w}_{k+1} = \mathbf{w}_k - \alpha \nabla g(\mathbf{w}_k)$.
- ◆ The local linear model predicts a decrease of about $\alpha \|\nabla g\|^2$ for a sufficiently small step.
- ◆ The gradient must be recomputed after every step.
- ◆ The learning rate α controls how far we move.
- ◆ A Python `for` loop lets us carry out repeated gradient-descent updates.



Next

- Gradient descent gives us a systematic learning rule.
- Its behavior depends strongly on the learning rate α .
- **1.6 - Step Size and Convergence**