

# Machine Learning

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

## Previous Lecture

- ♦ A cost function is a function of the model parameters.
- ♦  $\text{min}_g$  is the smallest cost;  $\text{argmin}_g$  gives the parameter that produces it.
- ♦ Learning can be written as  $\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}} g(\mathbf{w})$ .
- ♦ The graph or surface of  $g$  is the cost landscape.
- ♦ Python can evaluate and visualize the same optimization problem we derive mathematically.



## §2.3 thru 2.6 - Searching a Cost Function

Last time: learning became an optimization problem.

$$\mathbf{w}^* = \underset{\mathbf{w}}{\text{argmin}} g(\mathbf{w}).$$

But knowing the problem is not the same as solving it.

Today: search for a minimum using only values of  $g$ .

No derivatives yet.

## Zero-Order Search

Suppose we can evaluate  $g(\mathbf{w})$  at any parameter choice.

We do **not** use  $g'$  or  $\nabla g$ .

We only ask:  
which parameter choices give smaller cost?

Methods using only function values are called **zero-order methods**.

## Return to Our Cost Function

From Lesson 1.2:

$$f(x; w) = wx,$$

with data

|     |   |   |   |
|-----|---|---|---|
| $x$ | 1 | 2 | 3 |
| $y$ | 2 | 4 | 5 |

We found

$$g(w) = \frac{1}{3}(14w^2 - 50w + 45).$$

Exact minimizer:

$$w^* = \frac{25}{14} \approx 1.786.$$

Now pretend we do not know that answer.

## Grid Search

Choose a finite grid of possible values.

For example:

$$w = 0, 0.5, 1, 1.5, 2, 2.5, 3.$$

Evaluate the cost:

| $w$    | 0  | 0.5   | 1 | 1.5 | 2     | 2.5 | 3 |
|--------|----|-------|---|-----|-------|-----|---|
| $g(w)$ | 15 | 7.833 | 3 | 0.5 | 0.333 | 2.5 | 7 |

Smallest sampled cost occurs at

$$w = 2.$$

Close — but not exact.

## A Finer Grid

Use spacing 0.1 instead of 0.5.

Now the search includes

$$1.7, 1.8, 1.9, \dots$$

The best sampled value is

$$w = 1.8.$$

Finer grid  $\Rightarrow$  better approximation.

But also more evaluations of  $g$ .

## Grid Search in Python

Use the same cost function from last time:

```
def cost(w):
    y_pred = w*x
    return np.mean((y_pred - y)**2)
```

Build the grid:

```
w_values = np.linspace(0, 3, 7)
```

Evaluate every point:

```
cost_values = []
```

```
for w in w_values:
    cost_values.append(cost(w))
```

Find the smallest one:

```
best_index = np.argmin(cost_values)
w_values[best_index]
```

Output: 2.0.

New Python idea: `np` returns the location of the smallest entry.

---

## What Happens With Two Parameters?

Suppose

$$g = g(w_0, w_1).$$

Try 100 possible values for each parameter.

Number of grid points:

$$100 \times 100 = 10,000.$$

For three parameters:

$$100^3 = 1,000,000.$$

For ten parameters:

$$100^{10} = 10^{20}.$$

Grid search scales as

$$N^d,$$

where  $N$  = choices per parameter and  $d$  = number of parameters.

This becomes impossible very quickly.

---

## Random Search

Instead of testing a regular grid, choose parameter values randomly.

Example: choose 20 values of  $w$  between 0 and 3.

```
rng = np.random.default_rng(1)
w_values = rng.uniform(0, 3, 20)
```

Evaluate them:

```
cost_values = []

for w in w_values:
    cost_values.append(cost(w))

best_index = np.argmin(cost_values)
w_values[best_index]
```

Random search avoids a rigid grid.

But it can still miss the best region entirely.

---

# Global Search Versus Local Search

Grid search and random search can look across a large region.

Another idea: start somewhere and improve from there.

Choose a current point  $w$  and a step size  $\alpha > 0$ .

Compare:

$$g(w - \alpha), \quad g(w), \quad g(w + \alpha).$$

Move to whichever has the smallest cost.

This is a simple **local search**.

---

## A Local Search by Hand

Start with  $w = 0$  and step size  $\alpha = 0.5$ .

$$g(0) = 15.$$

Compare  $w = 0$  and  $w = 0.5$ .

$$g(0.5) = 7.833.$$

Move right.

Continue:

$$g(1) = 3,$$

$$g(1.5) = 0.5,$$

$$g(2) = 0.333,$$

$$g(2.5) = 2.5.$$

At  $w = 2$ , both nearby grid points are worse.

The search stops at  $w = 2$ .

Again: close, but limited by the step size.

---

## Coordinate Search

Now suppose  $\mathbf{w} = (w_1, w_2)$ .

Instead of moving everywhere at once, move along one coordinate at a time.

Coordinate directions:

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

From  $\mathbf{w}$ , compare

$\mathbf{w} \pm \alpha \mathbf{e}_1$   
and  
 $\mathbf{w} \pm \alpha \mathbf{e}_2$ .

Move in the coordinate direction that decreases  $g$  the most.

---

## Coordinate Search Example

Let

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

Start at

$$\mathbf{w}_0 = (0, 0),$$

with  $\alpha = 1$ .

Current cost:

$$g(0, 0) = 8.$$

Check the four coordinate moves:

$$\begin{aligned} g(1, 0) &= 5, \\ g(-1, 0) &= 13, \\ g(0, 1) &= 20, \\ g(0, -1) &= 4. \end{aligned}$$

Best move:

$$(0, 0) \rightarrow (0, -1).$$

Repeat from the new point.

---

## The Problem With Local Search

A local search sees only nearby values.

If the landscape has several valleys:

different starting points can lead to different answers.

A fixed step size can also be troublesome.

Too large  $\Rightarrow$  jump over a good point.

Too small  $\Rightarrow$  many evaluations.

Zero-order methods use little information about the shape of  $g$ .

---

## Can We Use More Information?

At a point  $\mathbf{w}$ , zero-order search asks:

“Which nearby point should I try?”

Calculus can answer a stronger question:

“Which direction makes  $g$  decrease?”

The derivative describes local change.

For many parameters, that information is contained in the **gradient**.

That is our next step.

---

## What did we learn?

- ◆ Zero-order methods search using values of the cost function only.
- ◆ Grid search becomes prohibitively expensive as the number of parameters grows.
- ◆ Random search samples the parameter space without using a fixed grid.
- ◆ Local and coordinate searches improve a parameter choice using nearby values.
- ◆ Derivatives can provide directional information that zero-order methods ignore.



---

## Next

- Zero-order methods can search a cost landscape.
- But they ignore how the cost is changing.
- **1.4 - The Gradient and Local Linear Approximation**