

Machine Learning

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

Previous Lecture

- ♦ Machine learning uses data to choose a model's parameters.
- ♦ Supervised learning uses input-output pairs (x_p, y_p) .
- ♦ A cost function measures how well a model fits the data.
- ♦ Learning often means minimizing that cost function.
- ♦ Python will implement and visualize the mathematics.



§1.4 thru 2.2 - Learning as an Optimization Problem

Last time: choose a model and measure its error.

Model: $f(x; \mathbf{w})$.

Cost: $g(\mathbf{w})$.

Learning: choose $\mathbf{w}$ so that $g(\mathbf{w})$ is small.

Today: look at that statement as a mathematical problem.

The Cost Is a Function

The inputs to g are the model parameters.

For a line:

$$f(x) = w_0 + w_1 x.$$

Then:

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

Change w_0 or $w_1 \Rightarrow$ change the model $\Rightarrow$ change the cost.

The data are fixed while we search over the parameters.

Optimization Vocabulary

Suppose $g(w)$ is a function of one parameter.

We want the parameter value where g is smallest.

Minimum value:

$$\min_w g(w).$$

Minimizer:

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

These are not the same thing.

Minimum Versus Minimizer

Example: $g(w) = (w - 3)^2 + 2$.

Smallest possible value:

$$\min_w g(w) = 2.$$

Where does it occur?

$$\underset{w}{\operatorname{argmin}} g(w) = 3.$$

Minimum = output of g .

Minimizer = input that produces it.

A One-Parameter Learning Problem

Suppose the data are:

x	1	2	3
y	2	4	5

Try models that are lines through the origin:

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

Only one parameter to learn: w .

Build the Cost Function

Mean squared error:

$$g(w) = \frac{1}{3} \sum_{p=1}^3 (wx_p - y_p)^2.$$

Substitute the data:

$$g(w) = \frac{1}{3} [(w - 2)^2 + (2w - 4)^2 + (3w - 5)^2].$$

Expand:

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

So learning has produced an ordinary function-minimization problem.

Find the Best Parameter — Without Calculus

Complete the square:

$$g(w) = \frac{14}{3} (w^2 - \frac{25}{7} w) + 15.$$

$$= \frac{14}{3} \left[\left(w - \frac{25}{14} \right)^2 - \left(\frac{25}{14} \right)^2 \right] + 15.$$

Therefore:

$$g(w) = \frac{14}{3} \left(w - \frac{25}{14} \right)^2 + \frac{5}{42}.$$

Thus:

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

and

$$\min_w g(w) = \frac{5}{42} \approx 0.119.$$

The data determined the parameter.

What Did We Actually Learn?

Our family of models was:

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

The learned model is:

$$f(x) = \frac{25}{14}x.$$

The optimization problem selected one member of the family.

Model family + data + cost + optimization

$\Rightarrow$ learned model.

The Cost Landscape

Think of g as a landscape over parameter space.

One parameter w :

$$\text{graph } y = g(w).$$

Two parameters (w_0, w_1) :

$$\text{surface } z = g(w_0, w_1).$$

Three or more parameters:

cannot draw the full landscape — but the optimization problem is the same.

Global and Local Minima

A **global minimizer** beats every allowable parameter choice.

$$g(w^*) \leq g(w) \text{ for every } w.$$

A **local minimizer** only beats nearby choices.

A cost function may have:

one minimum;

several local minima;

flat regions;
no attained minimum.

Optimization can be easy — or very difficult.

Python: Turn a Formula Into a Cost Function

Use the same data:

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

x = np.array([1, 2, 3])
y = np.array([2, 4, 5])
```

Define the model and cost:

```
def model(x, w):
    return w*x

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

Try a few values:

```
cost(1.5)
cost(2.0)
cost(2.5)
```

A Python function can represent the same $g(w)$ we derived by hand.

Python: Look at the Landscape

Create many possible parameter values:

```
w_values = np.linspace(0, 3, 100)
```

`linspace(0,3,100)` creates 100 evenly spaced numbers from 0 to 3.

Now evaluate the cost at each one:

```
cost_values = []

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

New Python idea: a **for loop** repeats a calculation.

Plot:

```
plt.plot(w_values, cost_values)
plt.xlabel("w")
plt.ylabel("cost g(w)")
plt.show()
```

We should see the minimum near $w = 1.786$.

Why Not Solve Every Problem Algebraically?

Here $g(w)$ was a quadratic.

Completing the square worked.

Real machine-learning models may have:

- thousands or millions of parameters;
- nonlinear cost functions;
- many local minima;
- no useful closed-form solution.

We need systematic numerical methods for searching a cost landscape.

What did we learn?

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



Next

- We now know what the optimization problem is.
- Next: how can we search for a minimum without using derivatives?
- **1.3 - Searching a Cost Function**