Week 2: Mathematical Foundations of Supervised Learning

Today's goals

By the end of this lesson, you should be able to

- understand supervised learning as an **optimisation problem**
- interpret **loss functions** as measures of prediction error
- understand how linear regression works
- recognise **overfitting** and **underfitting**

Question

Think first

How does a machine learning model know whether it is making good predictions?

Question

Think first

How does a machine learning model know whether it is making good predictions?

Discussion prompt

What should a model do when its predictions are too far from the real answers?

Supervised learning as optimisation

Main idea

In supervised learning, we choose a model and adjust its parameters so that its predictions become as accurate as possible.

Mathematical view

We want to find the best parameters by minimising an error measure.

$$\text{model} + \text{data} \rightarrow \text{loss} \rightarrow \text{minimum}$$

Key phrase

Training a model usually means **minimising loss**.

Data representation

Inputs as vectors

A single input can be written as a vector:

$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$

- each component is one feature
- examples could be height, age, temperature, or study time
- vectors are convenient for computation

Outputs: regression and classification

Regression	**Classification**
Predicts continuous values	Predicts discrete categories
Example: house price	Example: spam or not spam
Example: temperature tomorrow	Example: cat, dog, or bird

Main difference

Regression gives a number. Classification gives a class or label.

Linear models

Model form

A simple model can be written as

$$\hat{y} = wx + b$$

- x is just a scalar
- w is the **weight**
- b is the **bias**
- the model learns w and b from data

Interpretation

The weight controls the slope. The bias shifts the line up or down.

Loss functions

Idea

A loss function measures how wrong the model's predictions are.

Classic example, Mean Squared Error:

$$L = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

- y_i = real value
- $\hat{y}_i$ = predicted value
- the loss is small when predictions are close to the real answers

Why do we square the error?

Squared error

Using $(y_i - \hat{y}_i)^2$ has useful properties.

Why do we square the error?

Squared error

Using $(y_i - \hat{y}_i)^2$ has useful properties.

- negative and positive errors do not cancel out
- bigger mistakes are penalised more strongly
- the mathematics is often easier to work with

Important point

There are other loss functions too. Squared error is just one common choice.

Optimisation

Goal

Find the parameters that give the smallest possible loss.

$$\min_{w,b} L(w, b)$$

- we try different values of w and b
- the best values make predictions closer to the data
- optimisation is the process of searching for the minimum

Single-feature linear model

We started with

$$y = wx + b$$

where w is the weight and b is the bias.

Bundle the parameters into one vector

To write the model more compactly, we collect the learnable parameters into

$$\theta = \begin{bmatrix} w \\ b \end{bmatrix}$$

and use an augmented input

$$\tilde{x} = \begin{bmatrix} x \\ 1 \end{bmatrix}$$

so the same model becomes

$$y = \tilde{x}^\top \theta.$$

Gradients and learning

Main idea

A gradient tells us the direction in which the loss increases the fastest.

Gradients and learning

Main idea

A gradient tells us the direction in which the loss increases the fastest.

Question

The gradient with respect to what?

Gradients and learning

Main idea

A gradient tells us the direction in which the loss increases the fastest.

Question

The gradient with respect to what?

Why this helps

If we move in the opposite direction, we can reduce the loss.

Gradients and learning

Main idea

A gradient tells us the direction in which the loss increases the fastest.

Question

The gradient with respect to what?

Why this helps

If we move in the opposite direction, we can reduce the loss.

$$\text{new value} = \text{old value} - \text{step size} \times \text{gradient}$$

- gradients come from calculus
- the step size controls how big each move is

Optimisation in supervised learning

General recipe

We choose parameters θ to minimise an objective $J(\theta)$.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t)$$

- compute the gradient
- take a small step downhill
- repeat until the loss stops improving

Linear regression

Model

$$\hat{y}_i = x_i^\top \theta$$

Squared loss (Note, there is a factor of 1/2, does it make a difference, why is it there?)

$$\ell(\hat{y}_i, y_i) = \frac{1}{2}(\hat{y}_i - y_i)^2$$

Loss to be minimised (Objective)

$$L(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (x_i^\top \theta - y_i)^2$$

From sums to matrices

Start with the sum

$$L(\theta) = \frac{1}{2n} \sum_{i=1}^{n} (x_i^\top \theta - y_i)^2$$

Matrix notation

$$X\theta = \begin{bmatrix} x_1^\top \theta \\ x_2^\top \theta \\ \vdots \\ x_n^\top \theta \end{bmatrix}, \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Subtract

$$X\theta - y = \begin{bmatrix} x_1^\top \theta - y_1 \\ x_2^\top \theta - y_2 \\ \vdots \\ x_n^\top \theta - y_n \end{bmatrix}$$

Compact form (Different notation)

$$\sum_{i=1}^{n} (x_i^\top \theta - y_i)^2 = \|X\theta - y\|_2^2$$

Matrix form and solution

Stack the data

Let $X \in \mathbb{R}^{n \times d}$ and $y \in \mathbb{R}^n$.

$$L(\theta) = \frac{1}{2n} \|X\theta - y\|_2^2$$

$$\nabla_\theta L(\theta) = \frac{1}{n} X^\top (X\theta - y)$$

How?

To get the gradient as it is here, differentiate the sum on slide 15 and then change to the matrix notation. How does the sum interact with differentiation?

Set the gradient to zero

$$X^\top X\theta - X^\top y = 0$$

Question

Why can't you factorise and solve it the usual way? Can you think of two non-zero vectors that when multiplied equal zero?

If $X^\top X$ is invertible

This is the solution for linear regression in particular. When is $X^\top X$ invertible?

$$\theta^\star = (X^\top X)^{-1} X^\top y$$

Why algebra is useful

Computers need efficient formulas

Algebra lets us write model training in a way that is easier to compute.

- expressions can be simplified
- vectors and matrices help organise many numbers at once
- the same formula can handle many data points efficiently

Big idea

The maths helps turn learning into something a computer can calculate quickly.

Polynomial models

More flexible models

A linear model can be extended into a polynomial model.

$$\hat{y} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots$$

- can fit curved relationships
- still uses learned parameters
- matrices and vectors help when there are many features

Overfitting, underfitting, and bias–variance

Underfitting	**Overfitting**
Model is too simple	Model is too complex
Misses real patterns	Fits noise in the data
High training error	Very low training error
High **bias**, low variance	Low bias, high **variance**

Bias–variance tradeoff

Making a model more complex reduces bias but increases variance.

Goal

Find a balance so the model generalises well to new data.

Intuition

Overfitting

The model tries to follow every detail, including random noise.

Underfitting

The model is too simple to capture the real pattern.

Good fit

The model is complex enough to learn the pattern, but not so complex that it memorises noise.

Recap

What we learned today

- supervised learning can be seen as optimisation
- inputs are often represented as vectors
- regression predicts continuous values and classification predicts categories
- loss functions measure prediction error
- gradients help us improve models
- overfitting and underfitting describe poor model complexity choices

Next lesson

Unsupervised learning and Reinforcement Learning.