Lecture 1: Supervised Learning

Block 4.2 Advanced AI

Eoin O’Brien

Introduction

AI, machine learning, deep learning

- **Artificial intelligence (AI)**: systems intended to exhibit intelligent behaviour
  - includes logic, search, probabilistic reasoning and machine learning
- **Machine learning (ML)**: fits mathematical models to observed data
  - a subset of AI, not a synonym for it
- **Deep neural network**: one type of machine-learning model
- **Deep learning**: fitting deep neural networks to data
- Machine learning is commonly divided into
  - **supervised learning**
  - **unsupervised learning**
  - **reinforcement learning**

Supervised learning

Definition 1 A **supervised-learning model** maps an input to an output prediction and is fitted using observed input/output pairs.

- **Input**: the information available to the model
- **Target**: the output we want it to predict
- **Model**: a mathematical mapping from input to prediction
- **Training data**: labelled pairs $\{x_i, y_i\}$
- **Training / fitting**: choose a model from a family using those pairs
- **Inference**: apply the fitted model to an input to make a prediction

Regression and classification

Problem	Target	Output
regression	one continuous quantity	one number
multivariate regression	several continuous quantities	several numbers
binary classification	one of two categories	probabilities over two categories
multiclass classification	one of N categories	probabilities over N categories

- **Univariate regression**
  - one input
  - one continuous target
  - one predicted number

Inputs and outputs

- A real-world input is **encoded as numbers** before the model sees it
- The model maps numerical input to numerical output
- The output is then interpreted as a real-world prediction
- **Tabular**: fixed set of features
  - square footage, bedrooms, mass, temperature
  - reorder the features *and rebuild the model*: the meaning should not change
- **Ordered**: sequence matters
  - text: *my wife ate the chicken* $\neq$ *the chicken ate my wife*
- **High-dimensional / structured**
  - audio, images, molecules
  - many numbers, with important structure in how they are arranged

A ten-second audio clip sampled at $44.1\,\mathrm{kHz}$ contains **441,000 samples**.

Model family and training

- A model is a **mathematical equation** relating input to output
- Usually the equation contains **parameters**: numbers not fixed in advance
- Different parameter settings give different input/output relationships
- Together those relationships form a **model family**
- Training searches that family for a setting that describes the training pairs well

$$\mathbf{y} = \mathbf{f}[\mathbf{x}, \boldsymbol{\phi}] \qquad \hat{\boldsymbol{\phi}} \in \underset{\boldsymbol{\phi}}{\arg\min}\, \mathcal{L}[\boldsymbol{\phi}] \tag{1}$$

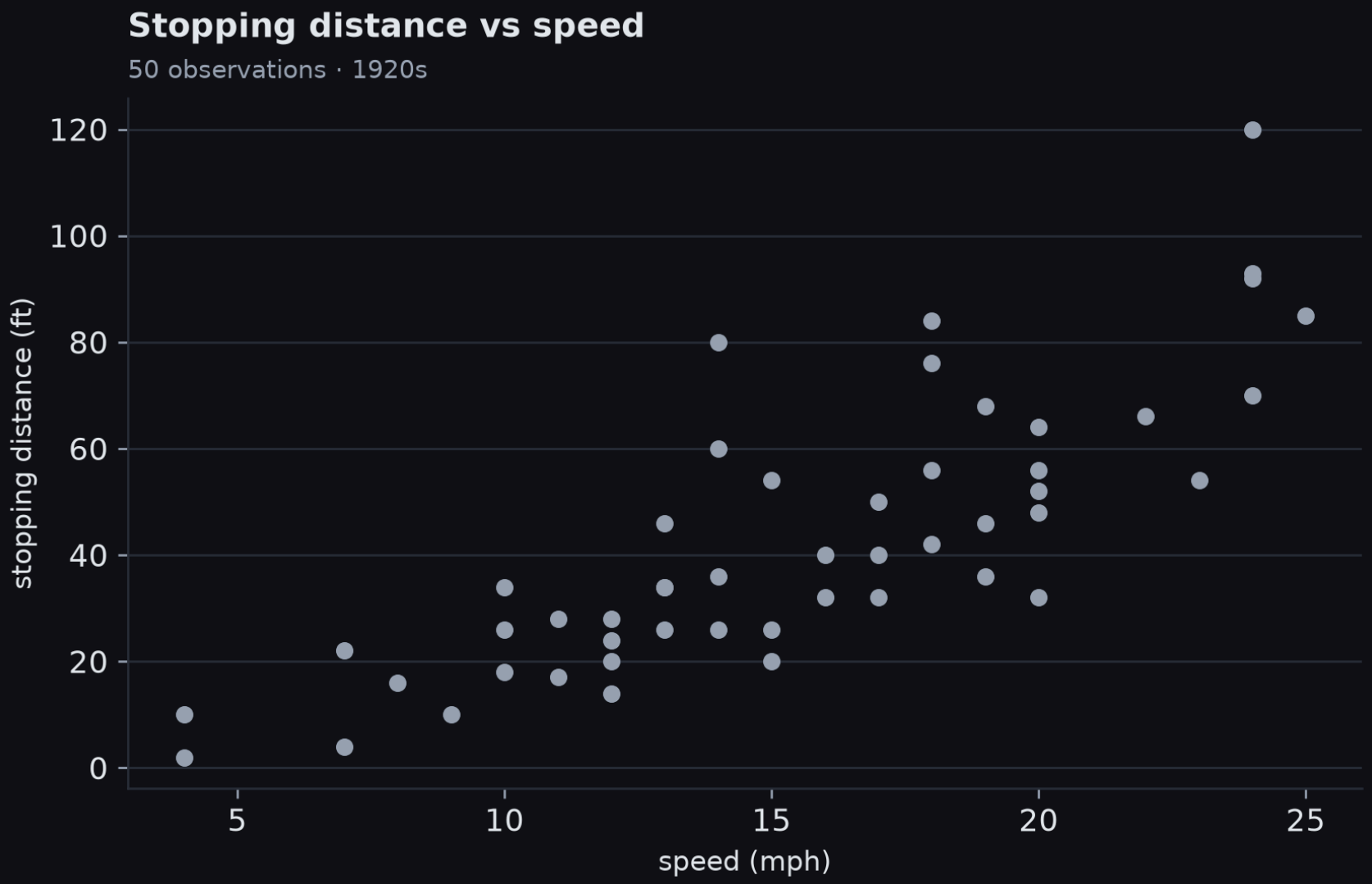
- $\mathbf{f}[\mathbf{x}, \boldsymbol{\phi}]$: parameters fixed, input varies — **inference**
- $\mathcal{L}[\boldsymbol{\phi}]$: data fixed, parameters vary — **training**

Notation

The book writes the parameter setting in **square brackets** and the ordinary function argument in **round brackets**. We keep that distinction.

Linear models

The dataset



Speed and stopping distance for 50 cars.

- **Input** x: the speed the car was travelling, in miles per hour
- **Target** y: the distance it took to stop, in feet
- 50 observations at 19 distinct speeds, recorded in the 1920s on 1920s cars (Ezekiel 1930)
- **Observation**: a pair in the *product* of the input and output spaces. The input space is the horizontal axis alone.

How can we model the relationship between speed and stopping distance? Could we use a line?

Slope and intercept

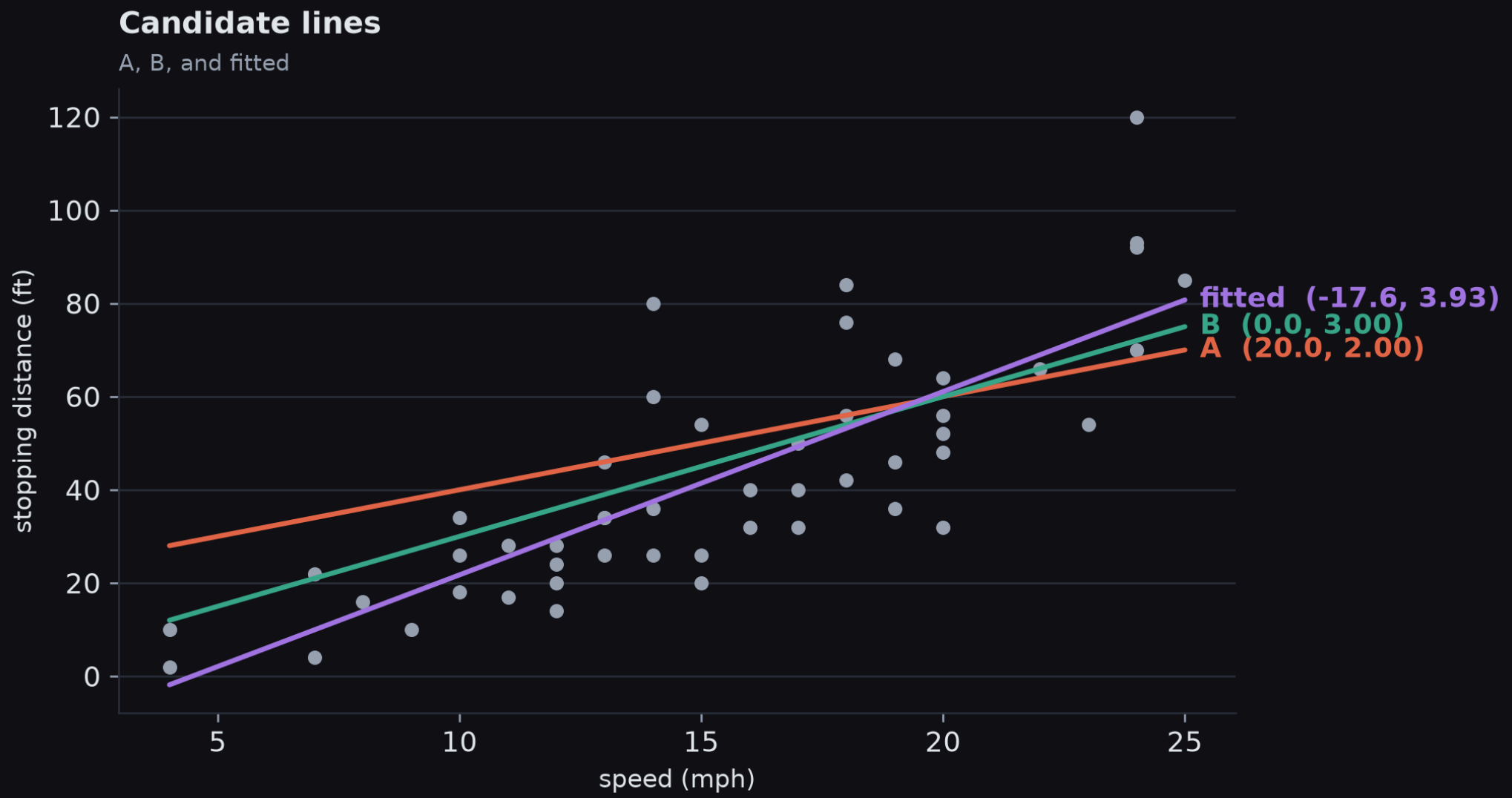
$$y = \mathrm{f}[x, \boldsymbol{\phi}] = \phi_0 + \phi_1 x \tag{2}$$

- **Model**: intercept plus slope times input
- **Shape**: two parameters, one input, one output
- **Units**
  - ϕ_0 is in **feet**
  - ϕ_1 is in **feet per mile per hour**
  - ...so the two parameters aren't interchangeable: neither their values nor their changes can be compared with each other

The fitted setting is $\phi_0 =$ -17.6 feet and $\phi_1 =$ 3.93 feet per mile per hour. A stationary car takes a *negative* distance to stop.

- At $x = 0$, the model is **extrapolating**
- The slowest observed car is doing 4 miles per hour
- No physical constraint is imposed at $x = 0$

Picking parameters



Three parameter settings on the same data.

Prediction, target, residual

- **Prediction**: the model's output for a given input, $\mathrm{f}[x_i, \boldsymbol{\phi}]$
- **Target** or **ground truth output**: the observed output for a given input, y_i
- **Residual** or **deviation**: the difference between the target and the prediction
  - Some texts give $y_i - \mathrm{f}[x_i, \boldsymbol{\phi}]$, others give $\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i$ (UDL does the latter)
  - Same magnitude, opposite sign
  - Squared in practice, so the sign doesn't matter
- **Shape**: one residual per $\{x_i, y_i\}$ pair, so a vector of I residuals

One car was doing 14 mph and took 80 ft. Under candidate B, $\phi_0 = 0$ and $\phi_1 = 3$. What is the residual, and what is its sign?

The prediction is 42 ft; the residual is +38 ft. The line sits *below* the observation.

From residuals to loss

How useful is a residual vector?

- Two candidate lines give two *vectors* of I residuals
  - Residual vectors have no natural total ordering
  - …so “which is better” has no default answer
- Searching for optimal parameters requires a way to compare them, so each residual vector must be reduced to **one score**
- We choose the convention that **smaller is better**
  - Fitting the model reduces to minimising that score

The scalar score is part of the model design.

Scoring residuals

- What if we just sum the residual vector?
  - Trivially poor: a badly wrong line can score zero if the residuals cancel
  - We want a criterion where the *direction* of a residual does not matter, only its size
- What if we sum the absolute values?
  - Better: the absolute value $|\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i| = |y_i - \mathrm{f}[x_i, \boldsymbol{\phi}]\,|$ is always non-negative
  - But one prediction off by 2 is treated the same as two predictions off by 1
  - Not differentiable everywhere, a big problem for gradient-based optimisation
- What if we sum the squares?
  - Better still: the square $(\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i)^2 = (y_i - \mathrm{f}[x_i, \boldsymbol{\phi}])^2$ is always non-negative
  - Penalises one large residual more than several small ones
  - Differentiable everywhere, so gradient-based optimisation works

Least-squares loss

Definition 2 The **least-squares loss** is the sum of the squares of the residuals over the training pairs.

$$\mathcal{L}[\boldsymbol{\phi}] = \sum_{i=1}^{I} \left(\mathrm{f}[x_i, \boldsymbol{\phi}] - y_i\right)^2 = \sum_{i=1}^{I} \left(\phi_0 + \phi_1 x_i - y_i\right)^2 \tag{3}$$

- **Shape**: I residuals in, one scalar out
- **Units**: ft². Squaring makes the units mathematically clear
  - ...but less directly interpretable than the residuals themselves.
- **Notation**: $\mathcal{L}$ is the quantity aggregated over the dataset; ℓ_i is one example's contribution
- Some sources distinguish between **loss** and **cost**:
  - A **loss function** calculates the loss for a single training example
  - A **cost function** aggregates the loss over the entire training dataset

Mean squared error

- Normalising the least-squares loss over the number of examples I yields the **mean squared error**
  - The advantage is that the MSE is independent of the dataset size
  - It can be compared across datasets meaningfully
- The two differ by a positive constant factor $\frac{1}{I}$

$$\mathrm{MSE}[\boldsymbol{\phi}] = \frac{1}{I}\mathcal{L}[\boldsymbol{\phi}] \tag{4}$$

- Multiplying a function by a *positive* constant leaves its **set of minimisers** unchanged and scales the **minimum value** by that constant
- The **argmin is unchanged**
  - The **minimum value** is scaled by I.
- Take care not to confuse the two!
  - Not numerically interchangeable unless the factor I is accounted for

Scoring two candidates

- **A**: $\phi_0 = 20$ feet, $\phi_1 = 2$ feet per mph
- **B**: $\phi_0 = 0$ feet, $\phi_1 = 3$ feet per mph
- Commit first: which scores lower?

Note that we can calculate the residuals either way; the squares will be identical:

x_i	y_i	$\phi_0 + \phi_1 x_i$	**residual**	**squared**
4	2	12	-10	100
4	10	12	-2	4
7	4	21	-17	289
7	22	21	+1	1

Sum the other 46 the same way, divide by 50: **A** scores 390.5 and **B** scores 261.3 ft^2.

B scores lower (better) **on this dataset**.

Loss as a function of parameters

- When working with loss, the **data is fixed** and the **parameters vary**.
  - During inference, the parameters are fixed and the data varies.
- So the loss is a function of the parameters, and we write $\mathcal{L}[\boldsymbol{\phi}]$

Object	Takes	Returns
the model $\mathrm{f}[x, \boldsymbol{\phi}]$	an input	a prediction
the loss $\mathcal{L}[\boldsymbol{\phi}]$	a parameter setting	one non-negative scalar

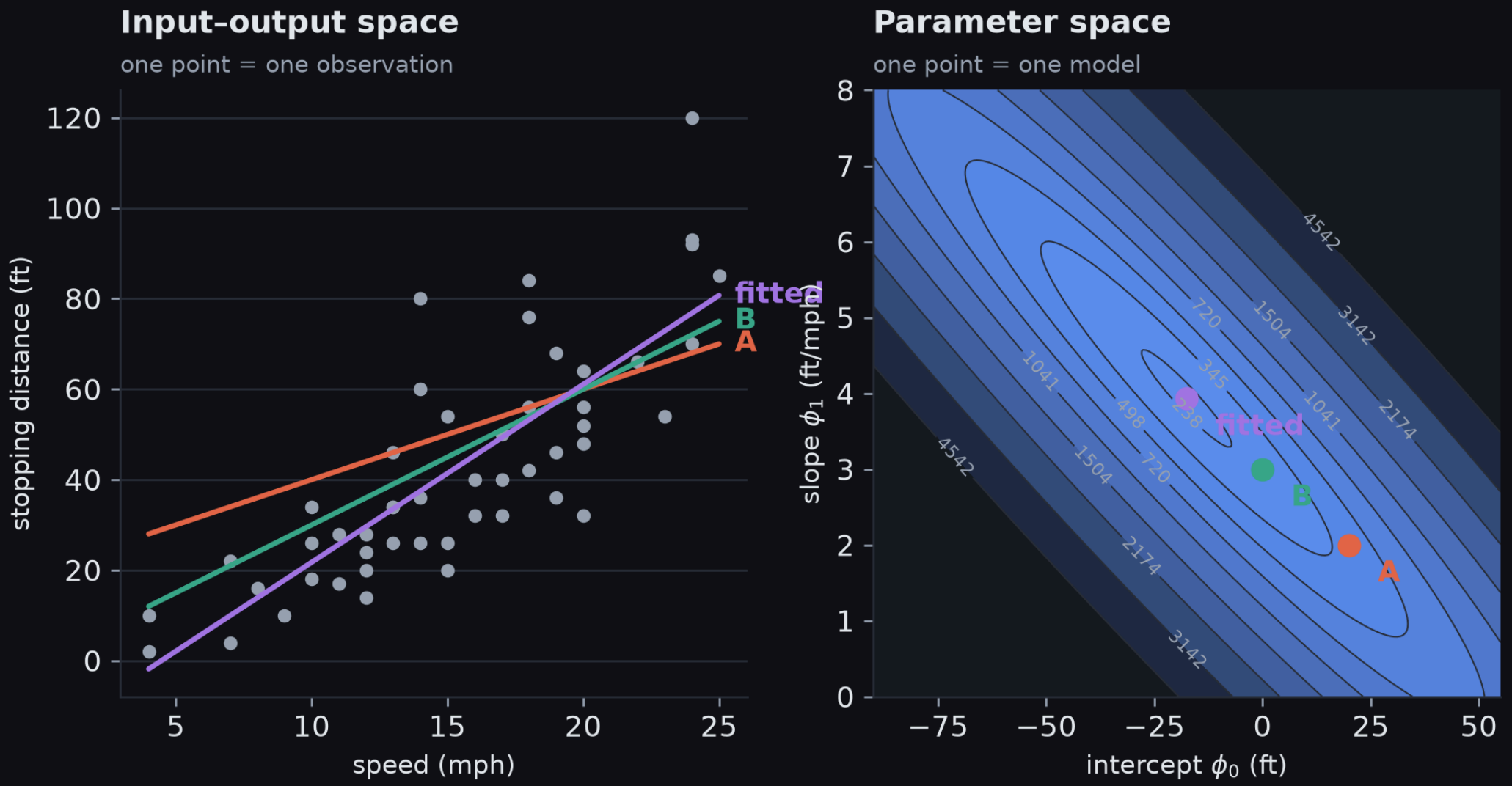
- Model and loss are **different functions of different variables**
- Here the loss takes **two parameters** and returns **one scalar**
- The data is held constant, so we don't have to write it as an argument

Data dependence

Add one more car and $\mathcal{L}$ becomes a different function! The notation suppresses the data, but the dependency remains.

Parameter space

Space exploration



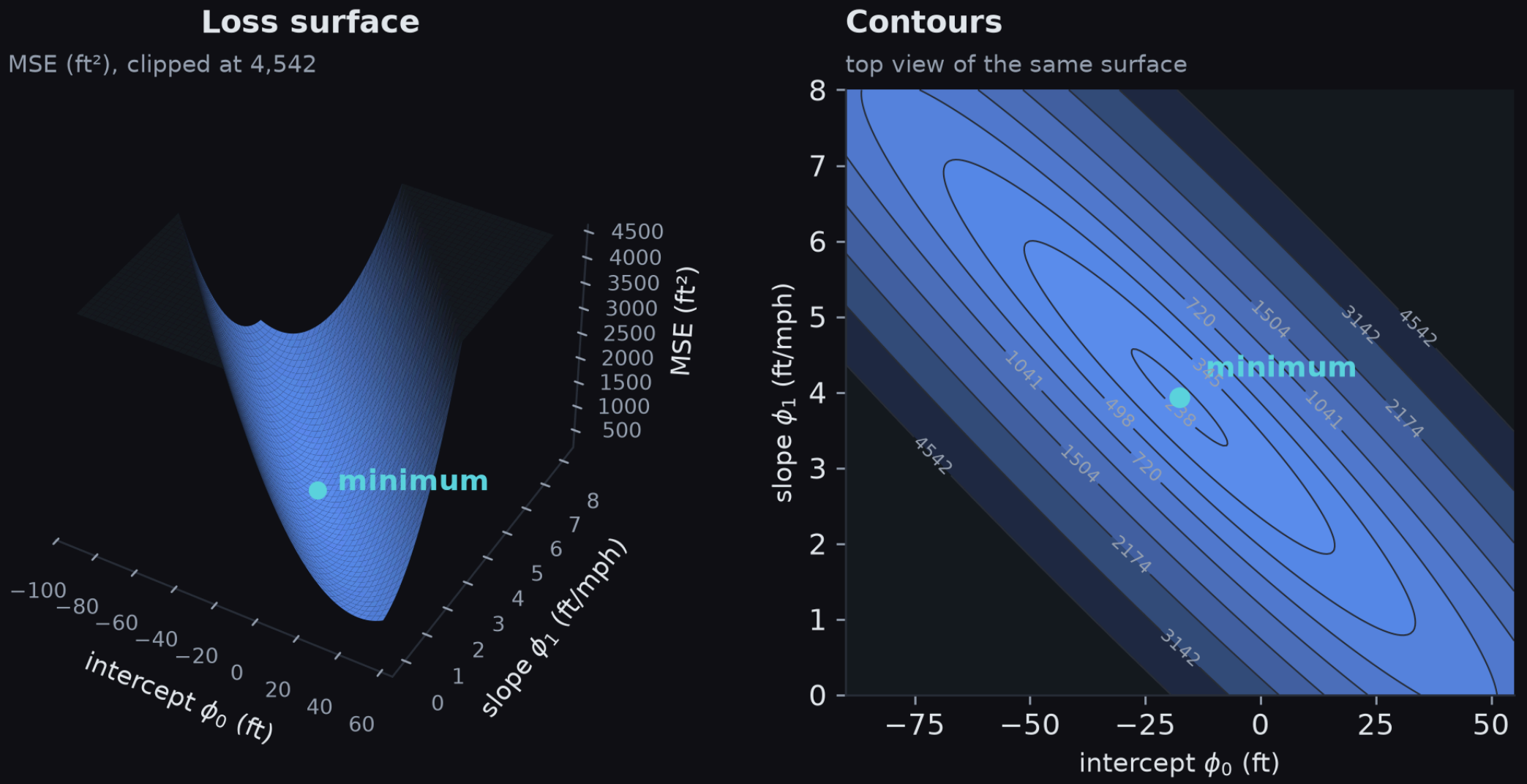
Candidate lines and their points in parameter space.

Space exploration (cont.)

Space	Dim.	One point	Where?
input	1	speed (mph)	left plot, x-axis
output	1	stopping distance (ft)	left plot, y-axis
input–output	2	observation (x, y)	left plot
parameter	2	one line/model setting	right plot

- One point in parameter space selects a model (line).
- The scatter plot is **not** a plot of the input space.
  - It's both input and output, so it's a plot of the **input–output space**.

Is this loss?

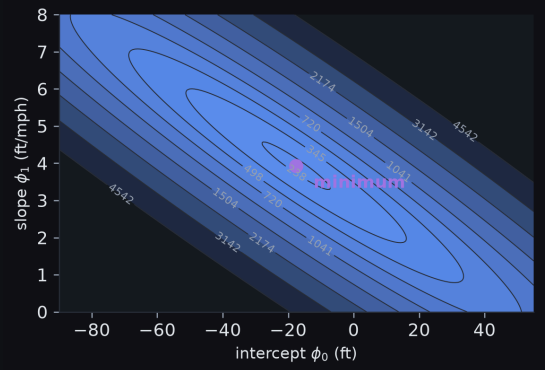


Mean squared error as a surface and as contours.

Is this loss? (cont.)

- The loss assigns one scalar to every point of the parameter plane
  - The left panel represents that scalar as **height**
  - The right panel shows the same function from above using contours
- **Shape**: a function from two dimensions to one
  - ...hence the 3-dimensional plot
- For this dataset, model family and loss, the surface has one lowest point

Reading the contours



Mean squared error contours in parameter space.

Definition 3 A **contour** is the set of parameter settings sharing one loss value.

- Contours packed closely means the loss changes quickly for a small move *in that direction*
- Elongated contours say the loss is more sensitive along some *directions* than others
  - they identify a particular *parameter* as more or less sensitive only when those directions line up with the coordinate axes
- Here the long direction is **tilted** relative to the parameter axes
  - move along it and a rise in the intercept trades against a fall in the slope
  - the fitted line changes much less than it would for a move across the contours

Min and argmin

- **min**: smallest loss value
  - a scalar, in ft² — 227.1 as a mean, or 11,354 as the chapter-2 sum
- **argmin**: parameter settings attaining that value
  - a subset of the parameter plane

$$\min_{\boldsymbol{\phi}}\left[\mathcal{L}[\boldsymbol{\phi}]\right] \in \mathbb{R} \qquad \arg\min_{\boldsymbol{\phi}}[\mathcal{L}[\boldsymbol{\phi}]] \subseteq \mathbb{R}^2 \tag{5}$$

- The text writes $\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}}[\mathcal{L}[\boldsymbol{\phi}]]$.
  - This assumes that the minimiser is unique (a common shorthand)
- Pedantically, it's a set, so $\hat{\boldsymbol{\phi}} \in \arg\min_{\boldsymbol{\phi}}[\mathcal{L}[\boldsymbol{\phi}]]$
  - The set of minimisers may have more than one element!
- **Non-unique minimum**: a flat valley would give one minimum value and infinitely many elements of the argmin

Fitting the line

Fitting

Definition 4 **Fitting** is choosing a parameter setting $\hat{\boldsymbol{\phi}}$ that minimises the loss over the training data, i.e. an element of the argmin.

$$\hat{\boldsymbol{\phi}} \in \underset{\boldsymbol{\phi}}{\arg\min} \left[\sum_{i=1}^{I} (\phi_0 + \phi_1 x_i - y_i)^2 \right] \tag{6}$$

- For this model and squared loss, the minimiser is unique **provided at least two input values differ**
  - if every input were identical, intercept and slope would trade off exactly
  - an entire line of the parameter plane would score the same
- With two distinct input values, the least-squares minimiser is unique.
- **Fitting**, **training** and **learning** are interchangeable terms for the same process

Finding the minimum

- So, how can we find the minimum?
- We could draw a grid over the parameter plane and sample every point
- Easy to draw, and all we have to do is calculate the loss for each point
  - Then pick the smallest!
- Is this a good idea?
- The grid returns the best point **on the grid**
  - Whether that's close to the best point on the plane is another story

Search	$\arg\min$	**min MSE (ft²)**
201-point grid	(-17.500, 3.920)	227.0872
exact	(-17.579, 3.932)	227.0704

- A **closed-form** solution also exists for this model.
  - We can't have nice things; in general, no closed-form solutions exist for more complex models
- There are much better ways to optimise!

Linear algebra

Scalars and scalar arithmetic

- A **scalar** is one real number: $a \in \mathbb{R}$
- Add or subtract scalars in the usual way
  - $a + b \in \mathbb{R}$
  - quantities being added must have compatible units (anyone do applied maths or physics?)
- Multiply scalars in the usual way
  - $ab \in \mathbb{R}$
- A scalar can scale a vector or matrix via **scalar multiplication**
  - $a\mathbf{x}$ for a vector, $a\mathbf{A}$ for a matrix

$$3 + (-1) = 2 \qquad 3(-1) = -3$$

- Technically, a scalar is an element of a field used to define a vector space
  - But for us, that field is always $\mathbb{R}$ and the vector space is always $\mathbb{R}^D$

Vectors: addition and scaling

- A vector in $\mathbb{R}^D$ is represented by D coordinates
  - written as a **column** unless stated otherwise
  - shape $(D \times 1)$
- **Vector addition**: same length, entry by entry
- **Scalar multiplication**: multiply every entry by the scalar

$$\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ -1 \\ 2 \end{bmatrix} = \begin{bmatrix} 5 \\ 1 \\ 5 \end{bmatrix} \qquad 2 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$$

- $\mathbf{x} + \mathbf{y}$ requires the **same shape**
- $a\mathbf{x}$ has the **same shape** as $\mathbf{x}$
- Adding a scalar to a vector is *not* a standard linear-algebra operation
  - Various libs will try to help by broadcasting the scalar over the vector and adding elementwise
  - Useful feature (no need to manually build a vector first) and potential footgun

The dot product: multiplying two vectors

Definition 5 For vectors of the same length, the **dot product** multiplies corresponding entries and adds the results.

$$\mathbf{a}^\top \mathbf{b} = \mathbf{a}^\top \mathbf{b} = \sum_{d=1}^{D} a_d b_d \tag{7}$$

For **non-zero** vectors:

- $\mathbf{a}^\top \mathbf{b} > 0$: angle $< 90^\circ$ — broadly aligned
- $\mathbf{a}^\top \mathbf{b} = 0$: angle $= 90^\circ$ — **orthogonal**
- $\mathbf{a}^\top \mathbf{b} < 0$: angle $> 90^\circ$ — broadly opposed

$$\mathbf{a}^\top \mathbf{b} = \|\mathbf{a}\| \ \|\mathbf{b}\| \cos\theta \tag{8}$$

- Result: **one scalar**
- Magnitude depends on both **length** and **alignment**
- Entrywise multiplication is a different operation, usually written $\mathbf{a} \odot \mathbf{b}$

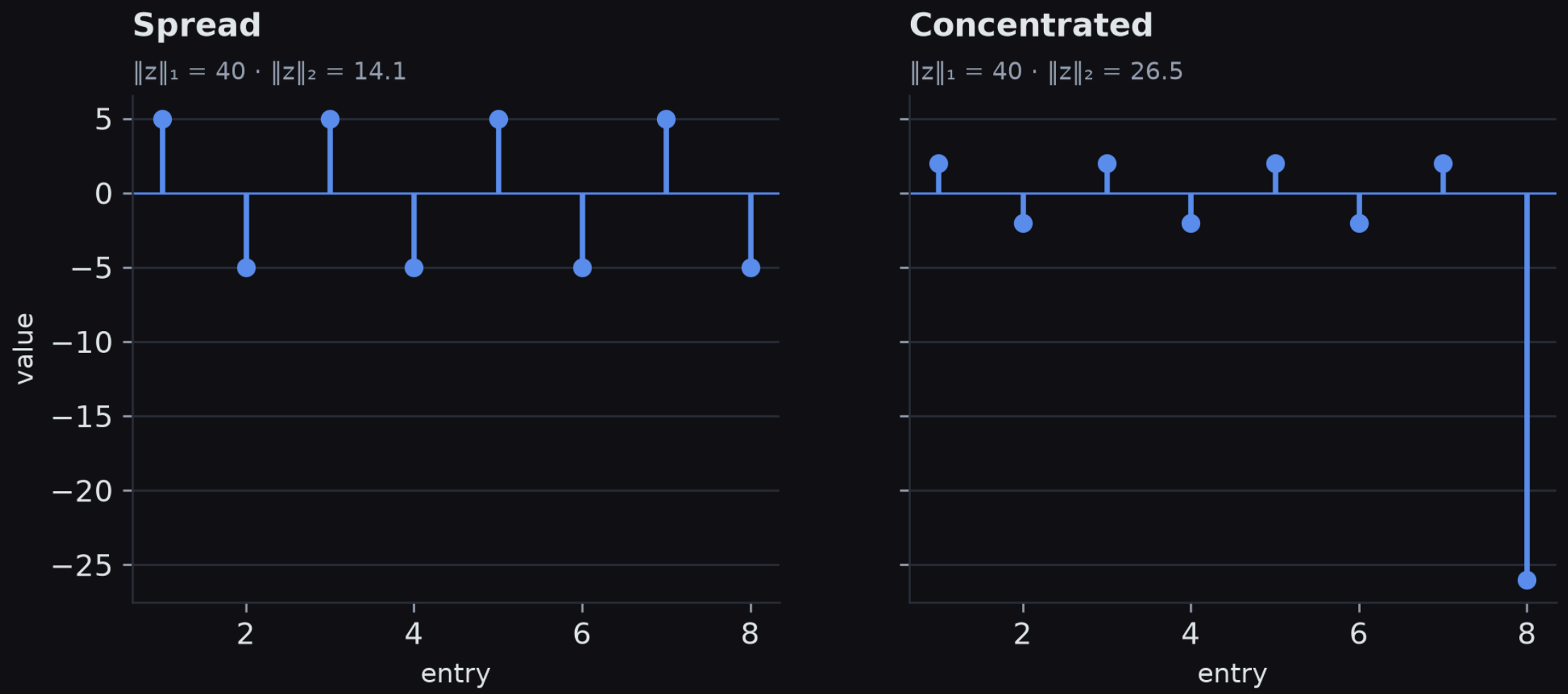
Norms

- A **norm** maps a vector to a non-negative scalar measuring its size

$$\|\mathbf{z}\|_p = \left(\sum_{d=1}^{D} |z_d|^p \right)^{1/p}, \qquad p \geq 1 \tag{9}$$

- $p = 1$: $\|\mathbf{z}\|_1 = \sum_d |z_d|$
- $p = 2$: Euclidean length, usually written $\|\mathbf{z}\|$
- Different norms can rank error patterns differently

Norms (cont.)



Same L1 norm; different L2 norm.

Matrices: addition and scaling

- A **matrix** is a rectangular array with shape $(D_1 \times D_2)$
- A_{ij}: row i, column j
- **Matrix addition**: same shape, entry by entry
- **Scalar multiplication**: multiply every entry
- **Transpose**: swap rows and columns

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix} \qquad 2\mathbf{A} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

- $\mathbf{A} + \mathbf{B}$ requires **identical shapes**
- $a\mathbf{A}$ keeps the shape of $\mathbf{A}$
- $\mathbf{A}^\top$ has shape $(D_2 \times D_1)$

Matrix–vector multiplication

- $\mathbf{A}$ has shape $(m \times n)$
- $\mathbf{x}$ has shape $(n \times 1)$
- $\mathbf{Ax}$ has shape $(m \times 1)$

Row view: each output is a dot product.

$$\mathbf{Ax} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} a_{11}x_1 + \cdots + a_{1n}x_n \\ \vdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n \end{bmatrix}$$

Each **row** of $\mathbf{A}$ produces one entry of the output.

Geometrically, column j of $\mathbf{A}$ is where the basis vector $\mathbf{e}_j$ is sent.

Matrix-matrix multiplication

- $\mathbf{A}$: $(m \times n)$
- $\mathbf{B}$: $(n \times p)$
- Inner dimensions agree, so $\mathbf{AB}$ is defined
- Result: $(m \times p)$

$$(m \times n)(n \times p) \to (m \times p) \qquad C_{ij} = \sum_{d=1}^{n} A_{id} B_{dj} \tag{10}$$

- Entry C_{ij} = row i of $\mathbf{A}$ dotted with column j of $\mathbf{B}$
- In general, $\mathbf{AB} \neq \mathbf{BA}$
- $(\mathbf{AB})^\top = \mathbf{B}^\top \mathbf{A}^\top$

Matching shapes

Defined? Resulting shape?

1. $(4 \times 3)(3 \times 5)$
2. $(5 \times 2)(5 \times 2)$
3. $(1 \times 7)(7 \times 1)$

What about the reverse?

Arithmetic at a glance

Expression	Requirement	Result
$a + b$	scalars	scalar
ab	scalars	scalar
$\mathbf{x} + \mathbf{y}$	same vector shape	vector
$a\mathbf{x}$	scalar and vector	vector
$\mathbf{x}^\top \mathbf{y}$	same vector length	scalar
$\mathbf{A} + \mathbf{B}$	same matrix shape	matrix
$a\mathbf{A}$	scalar and matrix	matrix
$\mathbf{A}\mathbf{x}$	inner dimensions agree	vector
$\mathbf{A}\mathbf{B}$	inner dimensions agree	matrix

Addition matches shapes. Dot and matrix products match inner dimensions.

Notation and shapes

Object	Written	Shape
scalar	x	one number
vector	$\mathbf{x}$	$(D \times 1)$ by default
matrix	$\mathbf{A}$	$(D_1 \times D_2)$
tensor	bold symbol	$D_1 \times \cdots \times D_N$

- Boldness carries type information
- PyTorch and TensorFlow use **tensor** for storage objects of many ranks
- Always track the mathematical **shape**, not just the library type

Maps, linear maps, affine maps

- A **map** $f : \mathcal{X} \to \mathcal{Y}$ sends each input to exactly one output
- A map is **linear** when

$$f(a\mathbf{u} + b\mathbf{v}) = af(\mathbf{u}) + bf(\mathbf{v})$$

{#def-linear-map}

- Linear maps preserve addition and scalar multiplication
- Therefore $f(\mathbf{0}) = \mathbf{0}$
- Between finite-dimensional vector spaces, a linear map has a matrix representation once bases are chosen

An **affine** map adds a translation:

$$f(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b} \tag{11}$$

- affine is linear **iff** $\mathbf{b} = \mathbf{0}$
- $y = mx$ is linear; $y = mx + c$ is affine

Span, independence, basis

- A **linear combination** of $\mathbf{v}_1, \ldots, \mathbf{v}_k$ is $a_1\mathbf{v}_1 + \cdots + a_k\mathbf{v}_k$
- The **span** is the set of all such combinations
- Vectors are **linearly independent** when no one is a linear combination of the others
- A **basis** is linearly independent and spans the space

$$\mathbf{e}_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{e}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{e}_3 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 3 \\ -1 \\ 4 \end{bmatrix} = 3\mathbf{e}_1 - \mathbf{e}_2 + 4\mathbf{e}_3 \tag{12}$$

The numbers are **coordinates in a basis**. Change the basis and the coordinates change; the vector does not.

Many inputs at once

More than one input

- Let's think back to our car dataset
- Is speed the only factor in stopping distance?
- What the 1920s recorders didn't write down
  - road surface, gradient, tyre condition, vehicle mass, driver reaction time
  - each could plausibly influence the stopping distance
- **More inputs**: one parameter per input, plus the intercept

$$y = \phi_0 + \phi_1 x_1 + \phi_2 x_2 + \cdots + \phi_D x_D \tag{13}$$

Indexing

Until now x_i meant example i. Here $x_1 \dots x_D$ are the D entries of **one** input. When both indices are needed, the example comes first: x_{id} is feature d of example i.

- Prediction, residual, loss and fitting keep the same definitions
- Writing the scalar expression out becomes cumbersome as the feature count grows, so the notation now has to carry the dimensions

The model as a dot product

- **Augmented input**: append a constant one and collect the intercept into the parameter vector

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ x \end{bmatrix} \qquad \boldsymbol{\phi} = \begin{bmatrix} \phi_0 \\ \phi_1 \end{bmatrix} \qquad \mathrm{f}[x, \boldsymbol{\phi}] = \tilde{\mathbf{x}}^\top \boldsymbol{\phi} \tag{14}$$

- This is the same model as Equation 2 from A SLOPE AND AN INTERCEPT, written in vector notation. No new family or capability has been introduced.
- **Shape check**: parameter vector of length $D + 1$, augmented input the same, output a scalar
- Keeping the bias separate gives the equivalent form $\mathbf{W}\mathbf{x} + \mathbf{b}$. PyTorch exposes these as `.weight` and `.bias`

Input vs parameters

The model is linear in the *augmented* input and remains affine in the original input. Appending a one is a bookkeeping convention, not a change of the map.

The design matrix

Definition 6 The **design matrix** stacks the augmented inputs, one example per **row**.

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_I \end{bmatrix} \qquad \mathbf{X}\boldsymbol{\phi} = \begin{bmatrix} \mathrm{f}[x_1, \boldsymbol{\phi}] \\ \mathrm{f}[x_2, \boldsymbol{\phi}] \\ \vdots \\ \mathrm{f}[x_I, \boldsymbol{\phi}] \end{bmatrix} \tag{15}$$

- **Shape**: I rows by $D + 1$ columns, and the column of ones is one of them
- **Predictions**: one per row, in a single product
- Which dimension disappears? $(I \times (D{+}1))((D{+}1) \times 1) \to (I \times 1)$. The parameter count is summed over; the example count survives.
- **Orientation**: rows are examples; columns are features. Transposing by accident is the most common shape bug in the first month.

The residual vector

Definition 7 The **residual vector** is the vector of targets minus the vector of predictions, one entry per example.

$$\mathbf{r} = \mathbf{y} - \mathbf{X}\boldsymbol{\phi} \tag{16}$$

- Entry i is exactly the residual you computed by hand at PREDICTION, TARGET, RESIDUAL, under the same sign convention
- This matches the statistics convention: $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$ and numerical linear algebra writes $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$. This is both of them.
- **Shape**: length I. It lives in a space with one dimension per **example**.

Space	Dimension here	One point is
input	1	a speed
parameter	2	a whole line over every input
residual	50	one model's mistakes, all of them

Three spaces are now in play. None of them substitutes for another.

One more feature

Linear in the input, linear in the parameters

- **Two notions of linearity**
  - is it linear as a function of the **input**, with the parameters fixed?
  - is it linear as a function of the **parameters**, with the input fixed?
- **Two-parameter line**: affine in the input; **linear in the parameters**
  - hold x fixed, and $\mathrm{f}[x, \boldsymbol{\phi}] = \tilde{\mathbf{x}}^\top \boldsymbol{\phi}$ is a dot product with $\boldsymbol{\phi}$ as the variable

$$\mathcal{L}[\boldsymbol{\phi}] = \sum_{i=1}^{I} \left(y_i - \tilde{\mathbf{x}}_i^\top \boldsymbol{\phi}\right)^2 \tag{17}$$

- With every x_i and y_i fixed:
  - each residual is **affine** in $\boldsymbol{\phi}$
  - each squared residual is **quadratic** in $\boldsymbol{\phi}$
  - their sum is quadratic with a positive-semidefinite quadratic term
- The loss is therefore **quadratic in the parameters**, giving the bowl shape and the closed-form least-squares solution.

Adding a squared term

- There is a physical reason to try a squared term, not merely a desire for more flexibility
  - stopping distance includes reaction distance and braking distance
  - under a simple braking model, braking distance grows approximately with the **square** of speed because kinetic energy grows with v^2
- Add a third design-matrix column holding x^2 and a parameter multiplying it

$$y = \phi_0 + \phi_1 x + \phi_2 x^2 \tag{18}$$

- **Family**: the two-parameter family is contained in it, at $\phi_2 = 0$.
- The new family cannot have higher **training** loss at its optimum: the old best is still available inside it at $\phi_2 = 0$
- Here the mean squared error falls from 227.1 to 216.5 ft² — an improvement of 4.7 per cent
- **Unchanged**: still a dot product, still linear in the parameters, still fitted by the same argmin, still a quadratic loss
  - the argmin itself **moves**, and gains a coordinate: it's a point in three-dimensional parameter space now

Parameter space gains a dimension

- Parameter space is now **three-dimensional**
- The loss is a map from three dimensions to one
- Its full graph would need **four**, so THE LOSS SURFACE's picture isn't available for this model

Still drawable	What it shows	What it hides
a 2-D slice, ϕ_2 held fixed	the surface at one setting of the third parameter	everything at every other setting
a projection	the range of loss values	which setting produced each
an isosurface	one loss level in 3-D	the values between levels

- The picture was a **teaching device tied to two parameters**. The mathematics never depended on it.

The closed form

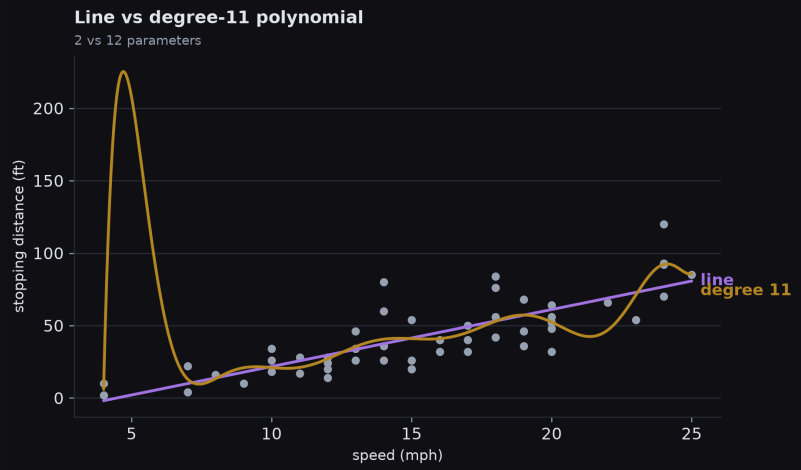
For a model **linear in its parameters** under the **squared loss**, a least-squares minimiser satisfies the **normal equations**.

$$\mathbf{X}^\top \mathbf{X}\, \hat{\boldsymbol{\phi}} = \mathbf{X}^\top \mathbf{y} \tag{19}$$

- **Shape check**: $\mathbf{X}^\top \mathbf{X}$ is square, with side equal to the parameter count
- If $\mathbf{X}$ has **full column rank**, the minimiser is unique and the matrix $\mathbf{X}^\top \mathbf{X}$ is invertible
  - x^2 is a function of x and is perfectly allowed: no *weighted sum* of the other columns reproduces it
  - checking pairs isn't enough — the columns 1, x and $1 + x$ are pairwise unrelated by scaling and still dependent
  - duplicate a column: the solution set becomes a line — Definition 4's degeneracy in matrix form
- **Here**: 19 distinct speeds, so every polynomial design up to degree 18 has full column rank **as mathematics**

Normal equations: scope

Up to degree eleven



Line and degree-11 polynomial fitted to the same data.

- Powers of speed up to the eleventh give 12 parameters including the intercept
- Training mean squared error 227.1 for the line against 182.9 for the polynomial, in ft^2
- ***Nonlinear in the input; still linear in its twelve parameters.***
  - so the same closed form, the same argmin *operation* and the same quadratic loss shape all apply — over a twelve-dimensional parameter space, at a different minimiser
- **Not established**: that it is a better model. Only that it scores lower on the data it was fitted to.

Rescaling the input

- The raw monomial columns live on radically different numerical scales
  - the first column is all ones
  - speed to the eleventh reaches about 2.4×10^{15} at the fastest car

Basis	Numerical rank	Condition number	Training MSE (ft^2)	Prediction (ft)
raw powers of speed	8 of 12	5.1×10^{19}	188.8	-103 to 95
speed scaled to $[-1, 1]$	12 of 12	19,691	182.9	6 to 225

- The design matrix has full column rank **as mathematics** in either basis; what changes is its numerical conditioning
- In the raw basis, the numerical rank estimate drops by 4 columns and the computed fit has **higher** training loss even though both bases span the same polynomial family
  - that is evidence that the numerical procedure has not recovered the same least-squares solution

When enumeration stops working

Grid search, counted

- Put a uniform grid of k points on each of P parameter axes
- The exhaustive product grid contains k^P parameter settings

$$N = k^P \tag{20}$$

- Holding the per-axis resolution fixed makes the total count multiplicative across dimensions.

Before the table: write down your estimate for twelve parameters, at the resolution from HOW WOULD YOU ACTUALLY FIND IT?

parameters	evaluations at 201 points per axis
2	40,401
3	8,120,601
12	4.3×10^{27}

⚠ **Uniform-grid cost**

This is a property of a uniform product grid at fixed per-axis resolution. It's not a theorem about optimisation. Nor is it a claim that the

Two parameters, twelve parameters

parameters	evaluations	wall clock
2	40,401	0.1 seconds
3	8,120,601	19.0 seconds
12	4.3×10^{27}	3.2×10^{14} years

- Same per-axis resolution throughout, at a measured 2.3 microseconds per loss evaluation on this dataset
  - the exact figure belongs to this machine and doesn't matter: nothing survives 4.3×10^{27} evaluations at any per-evaluation cost you could plausibly reach
- The obstacle here is the **enumeration strategy**, not the existence of a solution: the polynomial optimum is available in closed form without a grid.
- **Resolution trade-off**: coarser grids cost less but resolve less

Why enumeration fails

- **Objective**: defined over the full twelve-dimensional parameter space
- **Bottleneck**: exhaustive search over a product grid
- **Information used**: loss values, not local direction
- A method using local directional information could avoid most grid points

> ⚠ **Local information**
>
> Using local directional information does not by itself guarantee convergence or a global minimum.

Closed form vs local search

	Closed form	Local search
requires	linear in parameters; squared loss	local directional information
additionally	full column rank for uniqueness; stable solver	method-specific conditions
cost	linear least-squares solve	iterative steps

- **Requirements**
  - *linearity in the parameters* is a property of the **model family**
  - *full column rank* is a property of the **design matrix** — the data and the features you chose together
  - a **mathematical** rank failure requires changing the columns or accepting non-uniqueness
  - poor **numerical conditioning** can often be mitigated by a better-scaled basis or a more stable solver; rescaling helped in this example
- Break linearity in the parameters and this **linear least-squares** route no longer applies

What breaks parameter linearity?

- The polynomial family got substantially more expressive
- It never stopped being **linear in its parameters**

What would have to change about the model for that to stop being true?

What the loss is an estimate of

Empirical risk

Definition 8 The **empirical risk** is the average loss over the observed examples.

$$\hat{R}[\boldsymbol{\phi}] = \frac{1}{I}\sum_{i=1}^{I} \ell_i[\boldsymbol{\phi}] \tag{21}$$

- For squared loss, empirical risk is the **mean squared error**.
- *Empirical* means **over the data we have**
- Averaging gives a **per-example** quantity, on the same scale as expected loss

Population risk

Definition 9 The **population risk** is the expected loss over a specified distribution of input-output pairs — ideally the distribution whose performance we care about.

$$R[\boldsymbol{\phi}] = \mathbb{E}_{(x,y)\sim p}[\ell[\boldsymbol{\phi}; x, y]] \tag{22}$$

- **Meaning**: expected per-example loss over pairs drawn from p
- $\ell_i[\boldsymbol{\phi}]$ was $\ell[\boldsymbol{\phi}; x_i, y_i]$ all along — the per-example loss at one observed pair. Here the pair is drawn from p instead of being fixed at an observed example.
- $\hat{R}$ averages over the examples we have; R averages over the specified distribution p
- To use the training sample to estimate this quantity, we need a sampling assumption connecting those examples to p — commonly IID draws in the basic treatment
- **Shape**: both risks are single scalars, and both are functions of the parameters
- $\hat{R}$ is directly computable from the sample. R usually is not, because the full distribution p is not available.

Estimate and objective

- For a **fixed parameter setting chosen independently of the sample**, $\hat{R}[\phi]$ is an estimator of $R[\phi]$ under the sampling assumptions just stated
- During training we instead choose ϕ *because* it scores well on that same sample

That selection matters: training risk at the selected setting is generally an optimistic description of performance on new draws. Minimising empirical risk is therefore not the same thing as minimising population risk.

- Training curves show empirical risk on the data being used by the procedure
- Claims about performance must say which data and which distribution they refer to

Underfitting and overfitting

Definition 10 **Underfitting** means the fitted predictor performs poorly even on the training problem because the model family, optimisation or training procedure has not captured enough of the available structure.

Definition 11 **Overfitting** means performance on the training data is substantially better than performance on new data from the target distribution because the fitting procedure has adapted to sample-specific variation.

- Neither can be diagnosed from training loss alone
- Assessing generalisation requires data not used to fit the model

Not established

model	parameters	training MSE (ft^2)
line	2	227.1
line plus squared term	3	216.5
degree eleven	12	182.9

- Among these fitted models, the one with the most parameters has the lowest empirical risk
- That follows structurally because each larger family contains the previous one, so its optimum training loss cannot be higher
- **Not established**: that it predicts better, that it's the better model, or that more parameters help

Which of these would you ship, and what evidence would you need?

Consolidation

Retrieval

Closed notes, in pairs, then compared.

1. Reconstruct the difference between **input space** and **parameter space**, including the dimension of each for the two-parameter line — then name the space the scatter plot is a picture of, and its dimension.
2. State what **min** and **argmin** each return, with types.
3. One sentence: what does the loss take as its argument, and what is held fixed?

Review the item you could not reconstruct.

Reading

- **Prince, chapters 1 and 2**: the supervised-learning frame, model families, fitting and loss (Prince 2023)
- **Appendix A**: notation — square brackets, bold symbols and the conventions used throughout the major
- **Appendix B.3**: vectors, matrices and tensors — transpose, norms, matrix products and dot products used in the linear-algebra primer
    - use it as the slower reference if any part of the primer went past too quickly

Not required: appendix B.5, matrix calculus, and problems 2.1 and 2.2.

Two open questions

1. The polynomial became more expressive without ever becoming **nonlinear in its parameters**. What would change that?
2. Exhaustive search stopped scaling almost immediately, and the closed form won't always be there. What replaces them?

References

Ezekiel, Mordecai. 1930. *Methods of Correlation Analysis*. Wiley.

Prince, Simon J. D. 2023. *Understanding Deep Learning*. MIT Press. https://udlbook.github.io/udlbook/.

Questions?

Eoin O’Brien · eoin@eoin.ai