Mathematical Language of Deep Learning

Mathematics of Deep Learning — Chapter 1

Yin Yang

Foundation Books Project

Roadmap

Why a Mathematical Language?

The Running Workflow

Parameterized Functions

Data, Labels, and Shapes

Linear Algebra for Shape Reasoning

Predictions, Loss, Empirical Risk

Parameters, Representations, Layers

Training as Optimization

Generalization, Capacity, Inductive Bias

Notation and Reading Guide

Summary

Why a Mathematical Language?

What This Chapter Adds

Most introductions to deep learning lead with software: load data, build a model, call `fit`, report accuracy.

- This chapter slows the workflow down and *names* the mathematical objects underneath.
- No prior ML or programming assumed — only patient reading.
- Vocabulary: inputs, labels, shapes, parameterized functions, predictions, losses, empirical risk, gradients, generalization.

A precise vocabulary makes the rest of the book debuggable — shape errors and split errors become mathematical errors.

Why a Two-Feature Toy Example?

The chapter uses one tiny running example and keeps returning to it.

- Binary digit classifier: image $\rightarrow$ label in $\{0, 1\}$.
- Single example reduced to two features:

$$x = (x_1, x_2) = (0.20,\ 0.80).$$

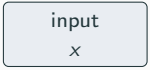
- Arithmetic stays small enough to do by hand.

Same chain of objects scales unchanged to MNIST, ImageNet, language models, retrieval, and generative models.

Small enough to inspect — structurally identical to the large models that follow.

The Running Workflow

The Running Workflow: Build It Up



Step 1. An example becomes numbers x.

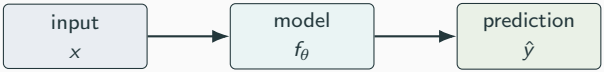
The Running Workflow: Build It Up



Step 1. An example becomes numbers x.

Step 2. The model f_θ produces output.

The Running Workflow: Build It Up



Step 1. An example becomes numbers x.
Step 2. The model f_θ produces output.
Step 3. That output is the prediction $\hat{y}$.

The Running Workflow: Build It Up



Step 1. An example becomes numbers x.
Step 2. The model f_θ produces output.
Step 3. That output is the prediction $\hat{y}$.
Step 4. A loss scores one prediction.

The Running Workflow: Build It Up



Step 1. An example becomes numbers x.
Step 2. The model f_θ produces output.
Step 3. That output is the prediction $\hat{y}$.
Step 4. A loss scores one prediction.
Step 5. Training changes θ to lower the loss.

Every later chapter elaborates one box in this picture.

First Numerical Pass

Two-feature score:

$$z = w_1 x_1 + w_2 x_2 + b.$$

With $x = (0.20, 0.80)$, $w_1 = 1.50$, $w_2 = -0.50$, $b = 0.10$:

$$z = 1.50(0.20) - 0.50(0.80) + 0.10 = 0.00.$$

- x_1, x_2: *observed*, fixed by the example.
- w_1, w_2, b: *parameters*, things training can change.
- z: a derived quantity — depends on both.

Inputs are observed. Parameters are chosen by training. The model output depends on both.

Parameterized Functions

From Function to Family

Function
$f : \mathcal{X} \to \mathcal{Y}$ assigns each input one output. $\mathcal{X}$ is the domain, $\mathcal{Y}$ is the codomain.

Parameterized model
$\{f_\theta : \mathcal{X} \to \mathcal{Y}\}_{\theta \in \Theta}$: a whole *family* of functions indexed by parameter θ.

Choosing θ picks one specific function from the family.

Training does not invent a function — it selects one from a pre-declared family.

Neural Network as Composition

A neural network is a composition of layer maps:

$$f_\theta = L_{m,\theta_m} \circ L_{m-1,\theta_{m-1}} \circ \cdots \circ L_{1,\theta_1}.$$



- Each layer has its own parameter block θ_j.
- "Deep" = many composed trainable stages.
- Composition is what makes the chain rule appear in training.

Three modeling choices must be checked before training: input information, output space, function family.

Data, Labels, and Shapes

Examples, Features, Labels

A dataset is a finite collection of examples:

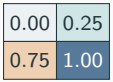
$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}.$$

- x_i: input for example i. y_i: target label.
- Features of example i: $x_i = (x_{i,1}, \ldots, x_{i,d})$.
- i indexes *examples*; j indexes *features*.

Confusing the two indices is a classic data-handling bug — "treating a feature coordinate as a separate example."

Always know which subscript names an example and which names a feature.

Flattening and Batching, Step by Step



2×2 image

(1) Image carries shape information.

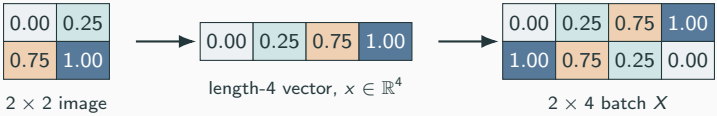
Flattening and Batching, Step by Step



(1) Image carries shape information.

(2) Flattening preserves values, changes shape only.

Flattening and Batching, Step by Step



(1) Image carries shape information.
(2) Flattening preserves values, changes shape only.
(3) A batch stacks m flattened examples as rows.

Shape errors are mathematical errors: objects no longer match the formula.

Numerical Scale Is Also a Choice

Pixel values are often integers in $\{0, 1, \ldots, 255\}$.

$$64/255 \approx 0.251.$$

Normalization to $[0, 1]$:

- does not change which pixels are darker or lighter,
- does change the numerical scale seen by the optimizer.

Other axes of choice: centering, channel statistics, integer vs. float, per-image vs. per-dataset normalization.

Data representation is already a mathematical choice with measurable training consequences.

Linear Algebra for Shape Reasoning

Three Operations That Cover Most Layers

Dot product
$w, x \in \mathbb{R}^d \Rightarrow w^\top x = \sum_j w_j x_j$ — a weighted sum, one scalar out.

Matrix-vector product
$W \in \mathbb{R}^{k \times d}$, $x \in \mathbb{R}^d \Rightarrow Wx \in \mathbb{R}^k$ — k weighted sums at once.

Affine map
$z = Wx + b$, with $W \in \mathbb{R}^{k \times d}$, $b \in \mathbb{R}^k$, output $z \in \mathbb{R}^k$.

Concrete check: $W \in \mathbb{R}^{4 \times 7}$, $x \in \mathbb{R}^7$, $b \in \mathbb{R}^4 \Rightarrow Wx + b \in \mathbb{R}^4$. A bias of length 7 would be a shape error.

Inner dimensions must match; outer dimensions give the result.

Batched Affine Map, Built Up

$$X$$
$$m \times d$$

(1) Batch matrix: one example per row.

Batched Affine Map, Built Up



(1) Batch matrix: one example per row.

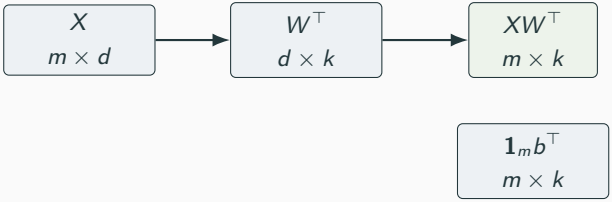
(2) Weights transposed so inner dimension d matches.

Batched Affine Map, Built Up



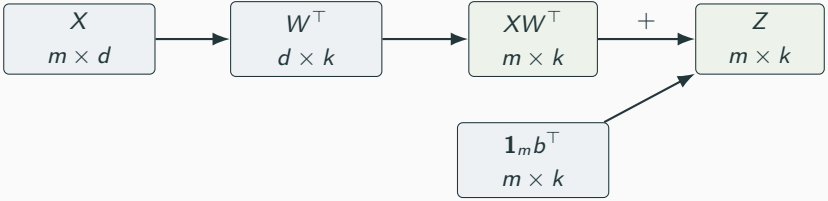
(1) Batch matrix: one example per row.
(2) Weights transposed so inner dimension d matches.
(3) Linear part $XW^\top$ has shape $m \times k$.

Batched Affine Map, Built Up



(1) Batch matrix: one example per row.
(2) Weights transposed so inner dimension d matches.
(3) Linear part $XW^\top$ has shape $m \times k$.
(4) Bias row is repeated across the batch.

Batched Affine Map, Built Up



(1) Batch matrix: one example per row.
(2) Weights transposed so inner dimension d matches.
(3) Linear part $XW^\top$ has shape $m \times k$.
(4) Bias row is repeated across the batch.
(5) Result Z: one k-vector of scores per example.

The batch axis m is preserved; the feature axis is transformed.

A Pocket Shape Table

Expression	Required shapes	Result shape
$w^\top x$	$w, x \in \mathbb{R}^d$	scalar
Wx	$W \in \mathbb{R}^{k\times d}$, $x \in \mathbb{R}^d$	$\mathbb{R}^k$
$Wx + b$	$W \in \mathbb{R}^{k\times d}$, $x \in \mathbb{R}^d$, $b \in \mathbb{R}^k$	$\mathbb{R}^k$
$XW^\top$	$X \in \mathbb{R}^{m\times d}$, $W \in \mathbb{R}^{k\times d}$	$\mathbb{R}^{m\times k}$
$W^\top g_z$	$W \in \mathbb{R}^{k\times d}$, $g_z \in \mathbb{R}^k$	$\mathbb{R}^d$
$g_z x^\top$	$g_z \in \mathbb{R}^k$, $x \in \mathbb{R}^d$	$\mathbb{R}^{k\times d}$
$\|x - y\|_2$	$x, y \in \mathbb{R}^d$	scalar

The last two rows already predict the shapes that backprop gradients must have — before any derivative is computed.

Shapes are a first-line check: if they don't match, no later arithmetic will save the formula.

Predictions, Loss, Empirical Risk

Prediction and Loss

Prediction
$\hat{y} = f_\theta(x)$. Read as "y hat" — model's output, distinct from the data target y.

Loss
$\ell(\hat{y}, y)$ is a scalar penalty. Smaller is better.

Squared error with $y = 1$:

Prediction	Loss	
$\hat{y} = 0.80$	$(0.80 - 1)^2 = 0.04$	close to target
$\hat{y} = 0.20$	$(0.20 - 1)^2 = 0.64$	far from target

The loss turns "predict well" into a number that training can minimize.

Empirical Risk = Average Loss

Given $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{n}$:

$$\widehat{R}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i).$$

Three example losses 0.04, 0.64, 0.09:

$$\widehat{R}(\theta) = \frac{0.04 + 0.64 + 0.09}{3} \approx 0.257.$$

The high middle loss visibly raises the average.

- Loss scores one prediction.
- Empirical risk scores the model on a sample.
- Accuracy is a different metric — two models can match in accuracy but differ in confidence.

State both what was optimized (loss) and what was reported (metric).

Parameters, Representations, Layers

Dense Layer Anatomy

A dense layer with input $x \in \mathbb{R}^d$ and output $h \in \mathbb{R}^k$:

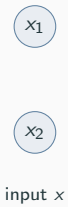
$$h = \sigma(Wx + b).$$

- $W \in \mathbb{R}^{k \times d}$: weight matrix. Parameter count: kd.
- $b \in \mathbb{R}^k$: bias vector. Parameter count: k.
- σ: activation, applied componentwise. Common: ReLU, sigmoid, softmax (at the output).

Example: $64 \to 10$ dense layer has $10 \cdot 64 + 10 = 650$ trainable parameters.

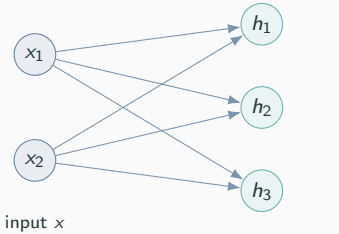
Parameter count is a property of the layer's shape, not of the data passing through it.

Composition Builds Representations, Step by Step



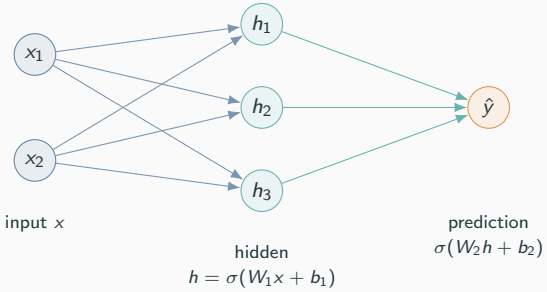
(1) Raw input features.

Composition Builds Representations, Step by Step



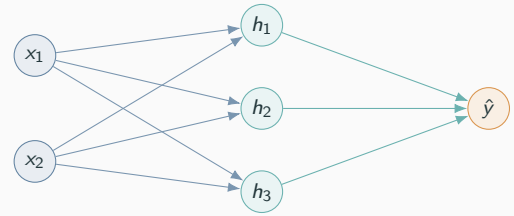
(1) Raw input features. **(2)** A hidden representation h is *learned*, not hand-designed.

Composition Builds Representations, Step by Step



(1) Raw input features. **(2)** A hidden representation h is *learned*, not hand-designed. **(3)** Final prediction reads off the representation.

Composition Builds Representations, Step by Step



Two affine layers without σ:
$W_2(W_1 x + b_1) + b_2 = (W_2 W_1)x + c$
collapses to one affine map.

(1) Raw input features. **(2)** A hidden representation h is *learned*, not hand-designed. **(3)** Final prediction reads off the representation. **(4)** Without nonlinearity, depth disappears.

Nonlinear activations are what keeps composition expressive.

Training as Optimization

Gradient and the Descent Direction

For a differentiable scalar $J(\theta)$, the gradient $\nabla_\theta J(\theta)$ is the vector of partial derivatives. It points toward steepest local increase.

Directional derivative in direction d:

$$\nabla_\theta J(\theta)^\top d.$$

Pick $d = -\nabla_\theta J(\theta)$:

$$\nabla_\theta J(\theta)^\top d = -\|\nabla_\theta J(\theta)\|^2 < 0.$$

A small step in $-\nabla_\theta J(\theta)$ decreases J.

The gradient is local information; "descent" is only a first-order guarantee.

Gradient Descent Step

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \widehat{R}(\theta_t), \qquad \eta > 0.$$

One-parameter example, $w_t = 0.50$, $\eta = 0.10$, $g = -0.30$:

$$w_{t+1} = 0.50 - 0.10(-0.30) = 0.53.$$

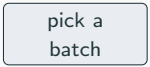
Tiny model: $\hat{y} = wx + b$, squared loss, $x = 2$, $y = 1$, $w = 0.20$, $b = 0.10$:

$$\hat{y} = 0.50, \quad \ell = 0.25, \quad \frac{\partial \ell}{\partial w} = 2x(wx + b - y) = -2.$$

Both $\partial \ell / \partial w$ and $\partial \ell / \partial b$ are negative, so the update raises $\hat{y}$ toward $y = 1$.

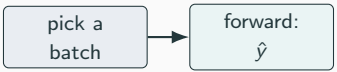
Learning rate is a step size — too small is slow, too large jumps past useful values.

One Training Step, End to End



(1) Choose a batch from $\mathcal{D}_{\text{train}}$.

One Training Step, End to End



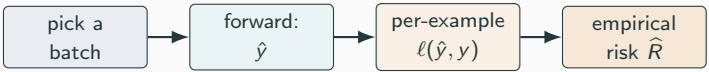
(1) Choose a batch from $\mathcal{D}_{\text{train}}$.
(2) Forward pass produces predictions.

One Training Step, End to End



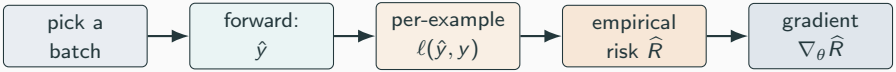
(1) Choose a batch from $\mathcal{D}_{\text{train}}$.
(2) Forward pass produces predictions.
(3) Per-example losses are computed.

One Training Step, End to End



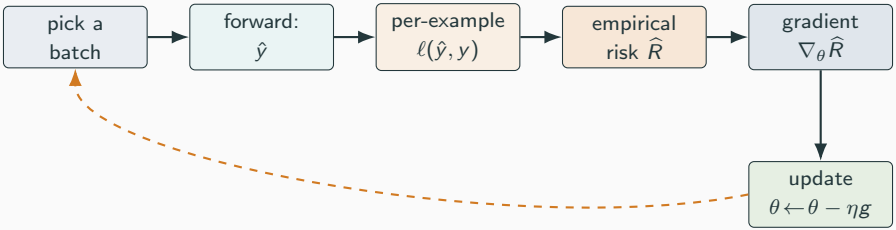
(1) Choose a batch from $\mathcal{D}_{\text{train}}$.
(2) Forward pass produces predictions.
(3) Per-example losses are computed.
(4) Average them: empirical risk on the batch.

One Training Step, End to End



(1) Choose a batch from $\mathcal{D}_{\text{train}}$.
(2) Forward pass produces predictions.
(3) Per-example losses are computed.
(4) Average them: empirical risk on the batch.
(5) Backprop computes $\nabla_\theta \widehat{R}$ efficiently.

One Training Step, End to End



(1) Choose a batch from $\mathcal{D}_{\text{train}}$.
(2) Forward pass produces predictions.
(3) Per-example losses are computed.
(4) Average them: empirical risk on the batch.
(5) Backprop computes $\nabla_\theta \widehat{R}$ efficiently.
(6) Apply the gradient step and repeat.

Backprop = chain rule, applied efficiently across the composed layers.

Generalization, Capacity, Inductive Bias

Two Risks, Two Samples

Population risk (the goal):

$$R(\theta) = \mathbb{E}_{(X,Y)\sim P}[\ell(f_\theta(X), Y)].$$

Empirical risk (what we can compute):

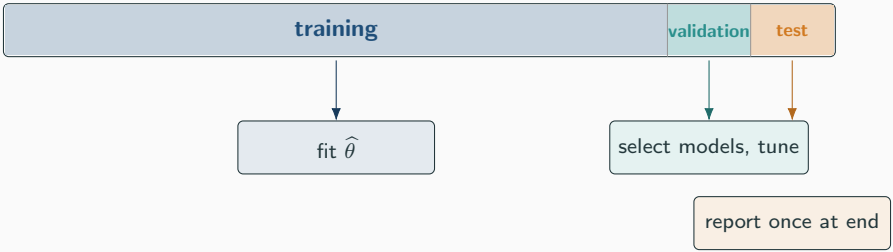
$$\widehat{R}_{\mathcal{D}}(\theta) = \frac{1}{n} \sum_{(x_i,y_i)\in\mathcal{D}} \ell(f_\theta(x_i), y_i).$$

Two snapshots of the same parameter θ:

Model	$\widehat{R}_{\text{train}}$	$\widehat{R}_{\text{val}}$
Small model	0.18	0.20
Very large model	0.02	0.64

Training loss is a sample average. The question is how it relates to a population expectation.

Train / Validation / Test — Different Roles



Generalization gap (formal): $\Delta(\theta) = R(\theta) - \widehat{R}_{\mathcal{D}}(\theta)$. Validation minus training is a computable *empirical proxy*, not the population gap itself.

Splits make the random nature of the fitted parameter visible.

Capacity and Inductive Bias

Hypothesis class: $\mathcal{F}_\Theta = \{f_\theta : \theta \in \Theta\}$.

- Larger $\Theta \Rightarrow$ lower achievable training risk.
- Larger $\Theta \Rightarrow$ more chances to fit sample noise.

Regularized empirical-risk minimization:

$$\widehat{\theta}_\lambda \in \arg\min_{\theta \in \Theta} \left\{ \widehat{R}_{\text{train}}(\theta) + \lambda\, \Omega(\theta) \right\}.$$

Inductive bias enters through the class, the parameterization, the penalty Ω, a prior, or the optimization algorithm itself (e.g. convolutions impose translation reuse).

"Does this model fit the task?" becomes "does this class and penalty make the right functions easy to find from the available sample?"

Notation and Reading Guide

Symbols by Role

Symbol	Meaning	Running-example interpretation
x	input	one digit image as numbers
y	label / target	0 or 1
$\hat{y}$	prediction	model's output for one image
f_θ	parameterized model	classifier with parameters θ
θ	parameter collection	all weights and biases
$\mathcal{D}$	dataset	labeled digit examples
$\ell(\hat{y}, y)$	loss	penalty for one prediction
$\widehat{R}(\theta)$	empirical risk	average loss on a dataset
∇_θ	gradient w.r.t. parameters	local training direction

Read each symbol by its type first: data, parameter, function, loss, or update direction.

Reading Habit: Formula $\leftrightarrow$ Workflow

Connect each formula to the corresponding workflow step:

Formula	Workflow meaning
$f_\theta(x)$	one forward prediction
$\ell(f_\theta(x_i), y_i)$	one example's penalty
$\widehat{R}(\theta)$	average penalty over a sample
$\nabla_\theta \widehat{R}(\theta)$	local information for an update
$\theta \leftarrow \theta - \eta \nabla_\theta \widehat{R}$	one training step

In code these appear as tensors, model calls, loss objects, and optimizer steps. The math gives those software objects a stable interpretation.

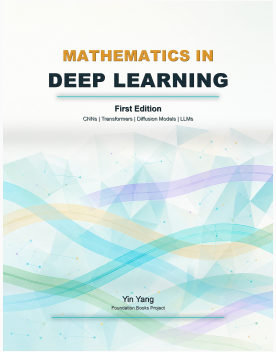
Notation is for removing ambiguity, not for ceremony.

Summary

Chapter in One Slide

- **Workflow:** data $\rightarrow$ model $\rightarrow$ prediction $\rightarrow$ loss $\rightarrow$ parameter update. Same picture from a toy classifier to a foundation model.
- **Functions:** deep learning chooses one f_θ from a declared family $\{f_\theta\}$; networks are compositions of layer maps.
- **Shapes:** most "data bugs" are shape mismatches. The pocket table catches them before any arithmetic.
- **Loss & risk:** loss scores one prediction; empirical risk averages over a sample.
- **Training:** follow $-\nabla_\theta \widehat{R}$ with step size η; backprop is just the chain rule, used efficiently.
- **Generalization:** train, validate, test are different *mathematical* roles, not just different filenames.
- **Inductive bias** is the part of the rule fixed before data is seen — class, penalty, parameterization, optimizer.

Next: probability, tensors, computational graphs, backpropagation, and the first practical training loops in code.



Mathematics in Deep Learning

Chapter overview slides for the book by Yin Yang.

Book page	`foundation-books.github.io/mathematics-in-deep-learning`
Code	`github.com/foundation-books/mathematics-in-deep-learning-code`
Paperback	`amazon.com/dp/B0H1ZZHDT2`
Hardcover	`amazon.com/dp/B0H1ZL11MJ`
Kindle	`amazon.com/dp/B0GX32M8SP`
License	CC BY-NC-ND 4.0

These free overview slides may be shared non-commercially with attribution and without modifications. Full explanations, exercises, and companion-code context are in the book and linked repository.