Functions, Decision Boundaries, and Neural Network Geometry

Mathematics of Deep Learning — Chapter 2

Yin Yang

Foundation Books Project

Roadmap

Running Example and Geometric View

Functions, Scores, and Decision Rules

Linear Boundaries in Two Dimensions

Probabilities, Activations, and Thresholds

When Linear Boundaries Fail

Feature Maps and Representation Geometry

Neurons, Layers, and Composition

Training as Moving a Boundary

First Backpropagation Example

Capacity, Overfitting, and Inductive Bias

Summary

Running Example and Geometric View

From Image to Two-Feature Geometry

The chapter views classification *geometrically*: a classifier partitions input space into decision regions.

- Running example: a 0-vs-1 digit classifier.
- Each image is summarized into two features:

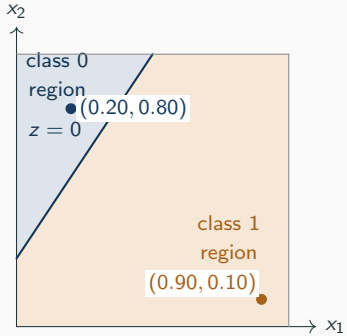
$$x = (x_1, x_2) \in \mathbb{R}^2, \qquad y \in \{0, 1\}.$$

- x_1 = vertical-stroke evidence, x_2 = closed-loop evidence.
- Running point: $x = (0.20, 0.80)$, true label $y = 0$.

Two features keep the geometry drawable and the arithmetic doable by hand.

Two features are a teaching simplification — the same geometry scales to high-dimensional tensors.

Decision Regions and Boundaries



- Decision region = set of inputs mapped to one predicted label.
- Decision boundary = where the predicted class can change.
- Crossing the boundary is the geometric event of changing a prediction.

A classifier is a partition of input space, not just a formula.

Functions, Scores, and Decision Rules

Three Objects, Three Roles

Scores, probabilities, and class labels are different mathematical objects.

Input and output spaces
$\mathcal{X} = \mathbb{R}^2$ (two-feature inputs); $\mathcal{Y} = \{0, 1\}$ (hard labels).

Score function
$s_\theta : \mathcal{X} \to \mathbb{R}$. The parameter θ collects everything training can change. Running example: $s_\theta(x) = w_1 x_1 + w_2 x_2 + b$.

Decision rule
Converts a score into a hard label, e.g. $\widehat{c}(x) = 1$ if $s_\theta(x) \geq 0$, else 0.

A score, a probability, and a class label are not the same object — name which one you are inspecting.

Score Rule on the Running Example

Weights $w_1 = 3$, $w_2 = -2$, bias $b = 0.5$.

Apply the score at $x = (0.20, 0.80)$:

$$s_\theta(x) = 3(0.20) - 2(0.80) + 0.5 = -0.5 \quad \Rightarrow \quad \widehat{c}(x) = 0.$$

At $x = (0.90, 0.10)$:

$$s_\theta(x) = 3(0.90) - 2(0.10) + 0.5 = 3.0 \quad \Rightarrow \quad \widehat{c}(x) = 1.$$

- Sign of s_θ chooses the side of the boundary.
- Magnitude $|s_\theta|$ measures how decisively, in score units.
- Threshold can be changed without changing the score function.

The threshold is part of the decision rule, not part of the function.

Linear Boundaries in Two Dimensions

Affine Scores and Their Boundary

Affine score
$a(x) = w_1 x_1 + w_2 x_2 + b$ with weight vector $w = (w_1, w_2)$ and bias b.

Decision boundary for threshold 0:

$$\{x \in \mathbb{R}^2 : \; w_1 x_1 + w_2 x_2 + b = 0\}.$$

When $w \neq 0$, this is a line; in $\mathbb{R}^d$ it is a hyperplane.

Running boundary $3x_1 - 2x_2 + 0.5 = 0$ solves to

$$x_2 = 1.5\, x_1 + 0.25,$$

a line with slope 1.5 and intercept 0.25.

Affine scores are the building block: dense layers, conv filters, and attention projections all start here.

Geometry of Weights and Bias

bias b
shifts line

(1) Changing b moves the line parallel to itself.

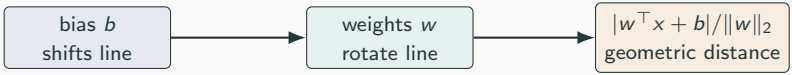
Geometry of Weights and Bias



(1) Changing b moves the line parallel to itself.

(2) Changing w_1, w_2 rotates the line and rescales the score.

Geometry of Weights and Bias



(1) Changing b moves the line parallel to itself.
(2) Changing w_1, w_2 rotates the line and rescales the score.
(3) Dividing by $\|w\|_2$ converts score to Euclidean distance.

Geometry of Weights and Bias



(1) Changing b moves the line parallel to itself.
(2) Changing w_1, w_2 rotates the line and rescales the score.
(3) Dividing by $\|w\|_2$ converts score to Euclidean distance.
(4) The sign of the score picks the half-plane.

For $w = (3, -2)$: $\|w\|_2 = \sqrt{13}$, so $x = (0.20, 0.80)$ has signed margin ≈ -0.139.

Weights set orientation, bias sets offset, norm gives geometric distance.

Probabilities, Activations, and Thresholds

From Score to Probability-Like Output

Sigmoid activation

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in (0, 1).$$

Mapping a score to a probability-like output:

$$\hat{y} = f_\theta(x) = \sigma(s_\theta(x)).$$

Examples for the running parameters:

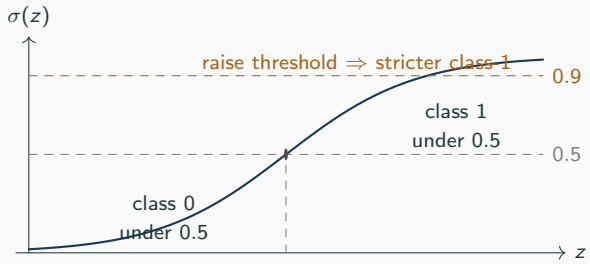
- $z = -0.5$: $\sigma(-0.5) \approx 0.378$.
- $z = \ \ 3.0$: $\sigma(3.0) \approx 0.953$.

Equivalence at the natural threshold:

$$\sigma(z) \geq \tfrac{1}{2} \iff z \geq 0.$$

"Probability-like" means right range, not necessarily calibrated.

Threshold Is a Decision Choice



- Medical screen: lower threshold to catch more positives.
- Costly action: higher threshold to require strong evidence.
- The *score function* did not change; the *rule* did.

Calibrate the threshold to the task, separately from training the score.

Activations Make Composition Nonlinear

Two common activations:

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \rho(t) = \mathrm{ReLU}(t) = \max(0, t).$$

- Sigmoid: smooth, $(0, 1)$-valued; classic output for binary probabilities.
- ReLU: piecewise-linear; common in hidden layers due to gradient behavior.
- Without a non-affine activation between layers, the entire stack collapses back to one affine map.

Debugging discipline: name which quantity is inspected — score z, output $\hat{y}$, or hard label $\widehat{c}$.

Nonlinear activations are what keep depth from collapsing.

When Linear Boundaries Fail

Linear Separability

Linearly separable (binary, in $\mathbb{R}^2$)
There exist w_1, w_2, b with $w_1 x_1 + w_2 x_2 + b > 0$ on every class-1 point and < 0 on every class-0 point.

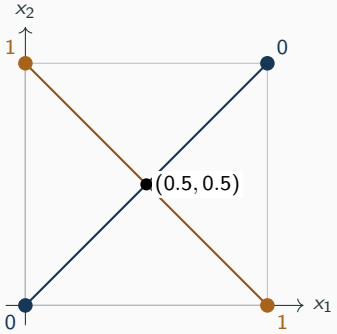
A strong requirement:

- One line, two half-planes, no exceptions allowed.
- Fails when classes are interleaved.
- Fails when useful information is hidden in a nonlinear feature interaction.

Practical signal: simple model has high error on *both* train and validation $\Rightarrow$ question the class, not the optimizer.

Underfitting is a geometry failure: the class cannot draw the needed shape.

XOR: A Concrete Failure



Both class-1 corners $(1, 0), (0, 1)$ and class-0 corners $(0, 0), (1, 1)$ share midpoint $(0.5, 0.5)$.

- Each side of a line is convex.
- Convex hull of class-1 pair includes the midpoint.
- Convex hull of class-0 pair includes the same midpoint.
- No single half-plane contains one without the other.

Feature Maps and Representation Geometry

Feature Maps Change the Coordinates

Feature map
A function $\phi : \mathcal{X} \to \mathcal{Z}$ replacing the input x with a representation $\phi(x)$ before classifying.

Apply a linear rule in feature space:

$$g(u) = a^\top u + c, \qquad u = \phi(x).$$

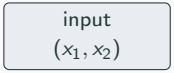
Resulting boundary back in input space:

$$\{x \in \mathcal{X} : \ a^\top \phi(x) + c = 0\}.$$

If ϕ is nonlinear, this need not be a hyperplane in the original (x_1, x_2)-plane.

Nonlinear work moves into the feature map; the final classifier can stay simple.

A Circle from a Linear Rule



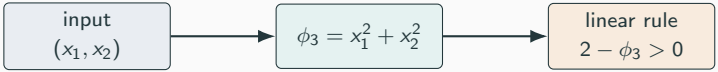
(1) Start in original (x_1, x_2) coordinates.

A Circle from a Linear Rule



(1) Start in original (x_1, x_2) coordinates.
(2) Append a nonlinear coordinate ϕ_3.

A Circle from a Linear Rule



(1) Start in original (x_1, x_2) coordinates.
(2) Append a nonlinear coordinate ϕ_3.
(3) A flat threshold on ϕ_3 is linear in feature space.

A Circle from a Linear Rule



(1) Start in original (x_1, x_2) coordinates.
(2) Append a nonlinear coordinate ϕ_3.
(3) A flat threshold on ϕ_3 is linear in feature space.
(4) Back in input space, the boundary is a circle.

Same arithmetic — one threshold — a curved decision in the original plane.

Why Representations Matter

Learned and pretrained representations:

- Embeddings encode similarity as distance in feature space.
- Transfer learning swaps the final linear head over a fixed encoder.
- Convolutions encode translation reuse before training begins.
- A small classifier on a strong representation often beats a large classifier on a weak one.

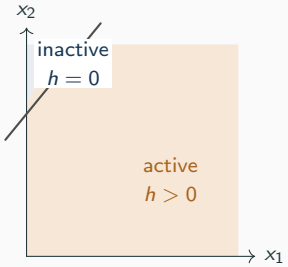
Carry forward: changing representations changes the coordinates in which a decision is made.

Most of deep learning is learning representations under which the final decision is simple.

Neurons, Layers, and Composition

A Neuron as a Half-Space Detector

Neuron
$h = \rho(w^\top x + b)$. ReLU case: $h = \max(0,\ w^\top x + b)$.



A ReLU unit is a learned detector for one side of a flat boundary — combine many such detectors to form a representation.

Each ReLU unit contributes one learned half-space test.

A Numerical ReLU Layer

Take $x = (0.20, 0.80)$ and

$$W = \begin{bmatrix} 3 & -2 \\ -1 & 2 \end{bmatrix}, \qquad b = \begin{bmatrix} 0.5 \\ -0.4 \end{bmatrix}.$$

Pre-activation:

$$Wx + b = \begin{bmatrix} 3(0.20) - 2(0.80) + 0.5 \\ -0.20 + 2(0.80) - 0.4 \end{bmatrix} = \begin{bmatrix} -0.5 \\ 1.0 \end{bmatrix}.$$

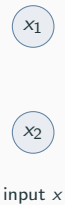
After componentwise ReLU:

$$h = \max(0,\ Wx + b) = \begin{bmatrix} 0 \\ 1.0 \end{bmatrix}.$$

Unit 1 is inactive on this input; unit 2 is active.

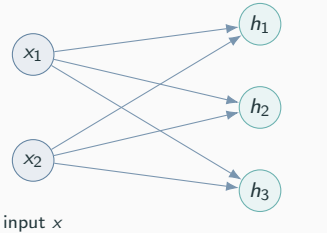
The hidden vector records which learned tests fired for this input.

Composition Builds Boundaries, Step by Step



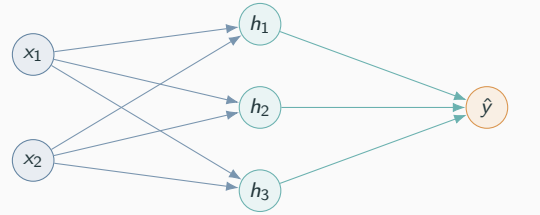
(1) Two input features.

Composition Builds Boundaries, Step by Step



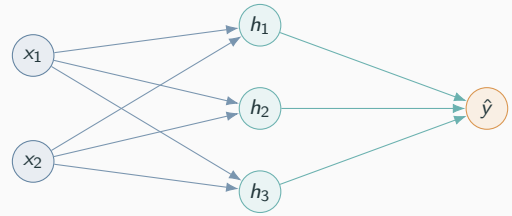
(1) Two input features. **(2)** Three learned half-space detectors.

Composition Builds Boundaries, Step by Step



(1) Two input features. **(2)** Three learned half-space detectors. **(3)** A simple readout over the hidden vector.

Composition Builds Boundaries, Step by Step



Without a nonlinearity:
$W_2(W_1x + b_1) + b_2 = (W_2W_1)x + d$
collapses to one affine map.

(1) Two input features. **(2)** Three learned half-space detectors. **(3)** A simple readout over the hidden vector. **(4)** Without ρ, depth disappears.

Depth helps only when each stage is allowed to bend the representation.

Training as Moving a Boundary

Loss and Gradient Descent for One Parameter

Squared error on one example:

$$L = (\hat{y} - y)^2.$$

Gradient descent update for parameter p:

$$p_{\text{new}} = p - \eta \frac{\partial L}{\partial p}, \qquad \eta > 0.$$

Sign convention:

- $\partial L/\partial p > 0 \Rightarrow$ descent decreases p.
- $\partial L/\partial p < 0 \Rightarrow$ descent increases p.

Learning rate η sets the step size along the local direction.

The gradient is local information — a nearby direction, not a global plan.

Moving the Running Boundary

Use the affine score as prediction: $\hat{y} = w_1 x_1 + w_2 x_2 + b$.

$$\frac{\partial L}{\partial b} = 2(\hat{y} - y).$$

At $x = (0.20, 0.80)$, $y = 0$, parameters $w_1 = 3$, $w_2 = -2$, $b = 0.5$:

$$\hat{y} = -0.5, \qquad \frac{\partial L}{\partial b} = 2(-0.5 - 0) = -1.$$

Gradient descent *increases* b. With $3x_1 - 2x_2 + b = 0$,

$$x_2 = 1.5\, x_1 + \tfrac{b}{2},$$

so the boundary line rises.

Whether the move helps depends on *all* labeled points — this is why training averages over batches.

Boundary Motion, Frame by Frame

batch
(x_i, y_i)

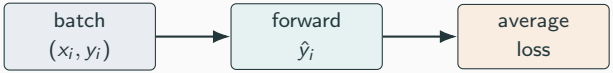
(1) Pick a batch of labeled examples.

Boundary Motion, Frame by Frame



(1) Pick a batch of labeled examples.
(2) Forward pass yields predictions.

Boundary Motion, Frame by Frame



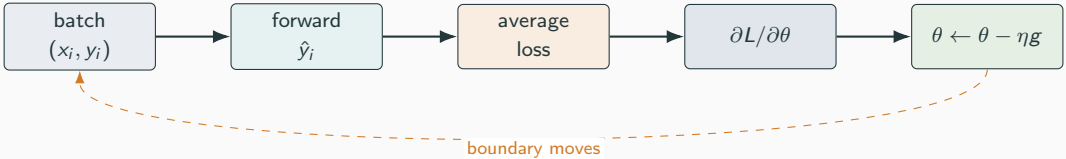
(1) Pick a batch of labeled examples.
(2) Forward pass yields predictions.
(3) Average loss summarizes batch error.

Boundary Motion, Frame by Frame



(1) Pick a batch of labeled examples.
(2) Forward pass yields predictions.
(3) Average loss summarizes batch error.
(4) Derivative gives a local direction.

Boundary Motion, Frame by Frame



(1) Pick a batch of labeled examples.
(2) Forward pass yields predictions.
(3) Average loss summarizes batch error.
(4) Derivative gives a local direction.
(5) Parameter update shifts the decision boundary.

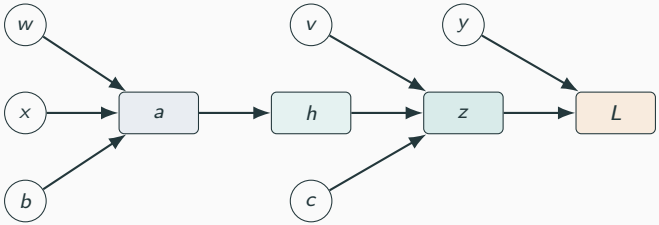
Training is repeated geometric editing of decision boundaries.

First Backpropagation Example

A Scalar Computational Graph

Toy chain:

$$a = wx + b, \quad h = a^2, \quad z = vh + c, \quad L = (z - y)^2.$$



Computational graph: nodes are values, edges record dependencies.

A neural network is a graph of differentiable scalar (and tensor) operations.

Chain Rule Along the Path

Chain rule for the path $w \to a \to h \to z \to L$:

$$\frac{dL}{dw} = \frac{dL}{dz}\,\frac{dz}{dh}\,\frac{dh}{da}\,\frac{da}{dw}.$$

Local derivatives:

$$\frac{dL}{dz} = 2(z-y), \quad \frac{dz}{dh} = v, \quad \frac{dh}{da} = 2a, \quad \frac{da}{dw} = x.$$

Numerical run with $x = 2$, $w = 0.5$, $b = 0$, $v = 3$, $c = 0.1$, $y = 4$:

$$a = 1, \quad h = 1, \quad z = 3.1, \quad L = 0.81,$$

$$\frac{dL}{dw} = 2(3.1-4)(3)(2)(2) = -21.6.$$

Gradient descent therefore increases w.

Backprop is bookkeeping for products of local derivatives along paths.

Why Products Matter

The path derivative is a *product* of local sensitivities.

- Many small factors ($\ll 1$): vanishing gradients — early parameters move slowly.
- Many large factors ($\gg 1$): exploding gradients — updates destabilize.
- Activation choice, initialization, and normalization all aim to keep these products well-behaved.

Later chapters lift this to:

- vector/matrix Jacobians,
- automatic differentiation,
- minibatch backprop on real architectures.

Stable training is partly a control problem on derivative products.

Capacity, Overfitting, and Inductive Bias

Capacity Is Boundary Flexibility

Model capacity
The range of partitions the class $\mathcal{H}$ can realize, before training picks one.

- Affine classifier in $\mathbb{R}^2$: one line, two half-planes.
- One-hidden-layer ReLU: piecewise-linear with many cells.
- Deep ReLU: layers can subdivide and reuse earlier regions.

Useful proxies: parameter count, hidden units, number of linear regions — none is a perfect summary.

Capacity is about a *class* of boundaries, not the one boundary you ended up with.

Three Failures, Different Cures

Class	$\widehat{R}_{\text{train}}$	$\widehat{R}_{\text{val}}$	Diagnosis
Single line	0.38	0.40	underfit (capacity too small)
Moderate ReLU network	0.07	0.10	balanced
Very flexible network	0.00	0.30	overfit (sample-specific fit)

- **Approximation error**: class cannot draw the right shape.
- **Estimation error**: finite sample misleads the selector.
- **Optimization error**: training fails to find the best member of the class.

Each gap names a different intervention — class, data, or optimizer.

Inductive Bias — Fixed Before the Data

Inductive bias for decision geometry
Any restriction or weighting of the allowed boundaries fixed before the training sample is observed.

Regularized empirical-risk minimization:

$$\widehat{h}_\lambda \in \arg\min_{h \in \mathcal{H}} \left\{ \widehat{R}_{\mathcal{D}}(h) + \lambda\, \Omega(h) \right\}.$$

Sources of bias:

- Function class (lines, polynomials, ReLU nets).
- Parameterization (sharing in convolutions, attention).
- Penalty $\Omega(h)$ (weight decay, sparsity).
- Optimization rule (SGD's implicit bias).

Pick the right bias and a large class becomes useful; ignore it and capacity alone overfits.

Universal Approximation, with a Caveat

Universal approximation (informal)
For many standard nonlinear activations, a feedforward network with enough hidden units can approximate any continuous function on a compact subset of $\mathbb{R}^d$ arbitrarily well.

What it does *not* say:

- How many units are needed in practice.
- How much data is required.
- Whether SGD will find the suitable parameters.
- Whether the learned function generalizes.

Capacity is paired with sample size, loss, and inductive bias — never considered alone.

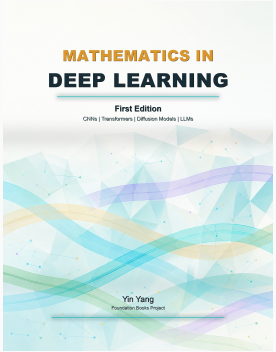
Existence in the class is not the same as findability from a finite sample.

Summary

Chapter in One Slide

- **Geometry:** a classifier partitions input space; boundaries are where predictions can change.
- **Scores, probabilities, labels:** three different objects — name which one you inspect.
- **Affine scores:** hyperplane boundaries; weights orient, bias shifts, $\|w\|$ rescales distance.
- **XOR:** convex half-planes cannot encode interleaved patterns.
- **Feature maps:** a linear rule in ϕ-space can be a curve in input space.
- **Neurons & layers:** ReLU units are learned half-space tests; nonlinearity keeps depth meaningful.
- **Training:** gradient descent moves boundaries; backprop computes the local direction efficiently.
- **Capacity & bias:** flexibility is paired with data; inductive bias is the part fixed before training.

Next: probability, tensors, computational graphs, and the matrix/batch version of everything that just happened scalar by scalar.



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.