Mathematics of Machine Learning and Large Language Models

MATH 76

Summer 2026

Contents

1 Introduction and the Postulates of Machine Learning

1.1 The Three Postulates

For the purposes of this course, machine learning is based on the following three postulates.

1. A state of a system admits a mathematical representation. The basic representation space will be a finite-dimensional vector space $\mathbb{R}^n$.
2. A machine can be represented as a function $f : \mathbb{R}^n \to \mathbb{R}^m$.
3. A learning machine has adjustable components. These components are represented by a parametrized family $\{f_\theta : \theta \in \Theta\}$, with $\theta = (\theta_1, \ldots, \theta_p) \in \Theta$.

Here Θ is the *parameter space*, and θ is a *parameter vector*. A supervised data set is a finite collection of ordered pairs $(x^{(i)}, y^{(i)})$, in which $x^{(i)}$ is an input and $y^{(i)}$ is its specified output. A *loss function* measures the discrepancy $\mathrm{dist}\big(\widehat{y}, y\big)$ between a predicted output $\widehat{y}$ and its specified output y. The empirical loss is, for example,

$$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \mathrm{dist}\big(f_\theta(x^{(i)}), y^{(i)}\big)$$

for a data set of size N. When L is differentiable, its gradient is $\nabla L = (\partial L/\partial\theta_1, \ldots, \partial L/\partial\theta_p)^{\mathsf{T}}$. A *gradient step* is the update $\theta \leftarrow \theta - \eta \nabla L(\theta)$, where $\eta > 0$ is the step size. These notes concern real, continuously varying parameters.

1.2 The First Parameter Families

The simplest parametrized families act on $\mathbb{R}^3$. In order of increasing richness, they are the following three, each an example of a parametrized family of functions.

- *Scalar scaling*, in which one number rescales the whole vector:

$$f_\lambda(x) = \lambda x, \qquad \theta = \lambda \in \mathbb{R}.$$

- *Coordinatewise scaling*, in which each coordinate carries its own factor:

$$f_\theta(x) = (\theta_1 x_1, \theta_2 x_2, \theta_3 x_3)^{\mathsf{T}}, \qquad \theta = (\theta_1, \theta_2, \theta_3) \in \mathbb{R}^3.$$

- The *general linear family*, in which an arbitrary matrix acts:

$$f_A(x) = Ax, \qquad A = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \\ \theta_{31} & \theta_{32} & \theta_{33} \end{bmatrix}.$$

These examples lead to the family of all linear maps, represented by matrices.

1.3 Vectors as Data and Geometry

1.3.1 Coordinates and Feature Vectors

A *feature vector* is a vector whose coordinates record prescribed numerical attributes of an object. We write a vector in $\mathbb{R}^d$ as

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix},$$

where each coordinate x_j is a real number. The same vector has two complementary interpretations.

Geometrically, x is a point in d-dimensional Euclidean space, where one may consider distances, angles, directions, and projections. As data, x is a list of features, and x_j is the value of the jth measurement or attribute. The Euclidean geometry of $\mathbb{R}^d$ thereby provides a means of comparing feature representations.

1.3.2 Vector Operations

The vector space $\mathbb{R}^d$ supports two operations: vector addition and scalar multiplication. If $x, y \in \mathbb{R}^d$ and $c \in \mathbb{R}$, then

$$x + y = \begin{bmatrix} x_1 + y_1 \\ \vdots \\ x_d + y_d \end{bmatrix}, \qquad cx = \begin{bmatrix} cx_1 \\ \vdots \\ cx_d \end{bmatrix}.$$

A *linear combination* of vectors $v_1, \ldots, v_k \in \mathbb{R}^d$ is an expression

$$c_1 v_1 + \cdots + c_k v_k,$$

where $c_1, \ldots, c_k \in \mathbb{R}$. The set of all such linear combinations is called the *span*:

$$\operatorname{span}\{v_1, \ldots, v_k\} = \{c_1 v_1 + \cdots + c_k v_k : c_i \in \mathbb{R}\}.$$

The standard basis vectors in $\mathbb{R}^d$ are

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad e_2 = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots, \quad e_d = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Every vector $x \in \mathbb{R}^d$ can be written uniquely as

$$x = x_1 e_1 + \cdots + x_d e_d.$$

Thus the coordinates of a vector are its coefficients relative to the chosen basis.

Remark 1.1. *Representations used in machine learning are generally coordinate-dependent. Pixel intensities and edge statistics, for example, determine different vectors. A transformed feature representation is often called an* embedding*; later, neural networks will produce embeddings from the original coordinates.*

1.3.3 Norms, Distances, and Inner Products

The Euclidean norm of $x \in \mathbb{R}^d$ is

$$\|x\|_2 = \sqrt{x_1^2 + \cdots + x_d^2}.$$

It measures the length of x. The Euclidean distance between x and y is

$$\|x - y\|_2 .$$

We shall often omit the subscript and write $\|x\|$ when the Euclidean norm is intended.

The inner product, or dot product, of $x, y \in \mathbb{R}^d$ is

$$\langle x, y \rangle = x^{\mathsf{T}} y = \sum_{j=1}^{d} x_j y_j.$$

The relation between the inner product and Euclidean geometry is

$$\langle x, y \rangle = \|x\| \, \|y\| \cos\theta,$$

where θ is the angle between the two nonzero vectors. Equivalently,

$$\theta = \arccos\!\left(\frac{x^{\mathsf{T}} y}{\|x\| \, \|y\|} \right).$$

For unit vectors this reduces to $\langle x, y \rangle = \cos\theta$. This number is called *cosine similarity*. The quantity $1 - \cos\theta$ is often called cosine dissimilarity; no assertion that it is a metric is intended.

Proposition 1.2 (Cauchy–Schwarz inequality)**.** *For all $x, y \in \mathbb{R}^d$,*

$$| \langle x, y \rangle | \le \|x\| \, \|y\| .$$

Equality holds if and only if one vector is a scalar multiple of the other.

Proof. If $y = \mathbf{0}$, the result is immediate. Otherwise, consider the function

$$f(t) = \|x - ty\|^2 = \|x\|^2 - 2t \langle x, y \rangle + t^2 \|y\|^2 .$$

Since $f(t) \ge 0$ for all real t, its discriminant is nonpositive:

$$4 \langle x, y \rangle^2 - 4 \|x\|^2 \|y\|^2 \le 0.$$

This gives $| \langle x, y \rangle | \le \|x\| \, \|y\|$. Equality occurs precisely when $x - ty = \mathbf{0}$ for some t, which means the two vectors are linearly dependent. □

For fixed w and fixed $\|x\|$, the Cauchy–Schwarz inequality shows that $| \langle w, x \rangle |$ is maximal precisely when x and w are linearly dependent. Thus a linear score has greatest magnitude on directions parallel or antiparallel to its weight vector.

1.3.4 Projection

Let u be a nonzero vector. The projection of x onto the line spanned by u is

$$\mathrm{proj}_u(x) = \frac{\langle x, u\rangle}{\langle u, u\rangle} u.$$

The scalar $\langle x, u\rangle / \langle u, u\rangle$ is the coordinate of the projection relative to the basis $\{u\}$ of $\mathrm{span}\{u\}$. The same operation appears in a linear score, which measures the component of an input in a prescribed direction.

Example 1.3 (Projection in $\mathbb{R}^2$)**.** *Let*

$$x = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \qquad u = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Then

$$\mathrm{proj}_u(x) = \frac{3 \cdot 1 + 1 \cdot 1}{1^2 + 1^2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}.$$

The residual is

$$x - \mathrm{proj}_u(x) = \begin{bmatrix} 1 \\ -1 \end{bmatrix},$$

which is orthogonal to u.

1.3.5 From Geometry to Scores

Given a nonzero vector $w \in \mathbb{R}^d$, the function

$$x \mapsto w^{\mathsf{T}} x$$

assigns a scalar score to every input vector. When w is a unit vector, $w^{\mathsf{T}} x$ is the coordinate of x along the axis in the direction w. The score is an inner product, so it measures alignment with the direction w. If w is fixed, all vectors satisfying

$$w^{\mathsf{T}} x = c$$

lie on a hyperplane. In $\mathbb{R}^2$ this is a line; in $\mathbb{R}^3$ it is a plane; in higher dimensions it is still called a hyperplane. The sets $\{x : w^{\mathsf{T}} x > c\}$ and $\{x : w^{\mathsf{T}} x < c\}$ are its two *open half-spaces*.

In binary linear classification, defined below, the score $w^{\mathsf{T}} x$ is compared with a threshold. The two open half-spaces determined by the corresponding hyperplane receive the two labels.

2 Linear Maps and Linear Layers

2.1 Matrices

An $m \times n$ matrix is a rectangular array of real numbers

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}.$$

We write $A \in \mathbb{R}^{m\times n}$. The entry in row i and column j is a_{ij}.

We shall use three descriptions of a matrix, corresponding to the three pictures of matrix-vector multiplication developed below.

1. The *entrywise picture* regards A as the array (a_{ij}).
2. The *column picture* regards A as an ordered collection of vectors in $\mathbb{R}^m$.
3. The *row picture* regards A as an ordered collection of row vectors, each of which defines an inner product with the input.

These descriptions encode the same linear map and differ only in which part of the multiplication is emphasized.

2.2 Matrix-Vector Multiplication

Let $A \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$. The product $Ax \in \mathbb{R}^m$ admits three equivalent descriptions.

The entrywise picture. The ith output coordinate is obtained by multiplying corresponding entries in row i of A and in x, and then adding:

$$(Ax)_i = \sum_{j=1}^{n} a_{ij}x_j.$$

Thus

$$Ax = \begin{bmatrix} \sum_{j=1}^{n} a_{1j}x_j \\ \sum_{j=1}^{n} a_{2j}x_j \\ \vdots \\ \sum_{j=1}^{n} a_{mj}x_j \end{bmatrix}.$$

The column picture. If $c_1, \ldots, c_n \in \mathbb{R}^m$ are the columns of A, then

$$A = \begin{bmatrix} c_1 & c_2 & \cdots & c_n \end{bmatrix}, \qquad Ax = x_1c_1 + \cdots + x_nc_n.$$

The input coordinates are the coefficients in a linear combination of the columns. This picture describes how output vectors are synthesized.

The row picture. If $r_1^{\mathsf{T}}, \ldots, r_m^{\mathsf{T}} \in \mathbb{R}^{1\times n}$ are the rows of A, then

$$A = \begin{bmatrix} r_1^{\mathsf{T}} \\ r_2^{\mathsf{T}} \\ \vdots \\ r_m^{\mathsf{T}} \end{bmatrix}, \qquad Ax = \begin{bmatrix} r_1^{\mathsf{T}}x \\ r_2^{\mathsf{T}}x \\ \vdots \\ r_m^{\mathsf{T}}x \end{bmatrix}.$$

Hence a matrix may be regarded as a collection of rows: row i takes its dot product with the input x and produces output coordinate i. This picture describes how scalar measurements are obtained from the input.

Example 2.1 (The three pictures)**.** *Let*

$$A = \begin{bmatrix} 1 & 3 \\ 2 & -1 \\ 0 & 4 \end{bmatrix}, \qquad x = \begin{bmatrix} 2 \\ -1 \end{bmatrix}.$$

The entrywise calculation is

$$Ax = \begin{bmatrix} 1(2) + 3(-1) \\ 2(2) + (-1)(-1) \\ 0(2) + 4(-1) \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \\ -4 \end{bmatrix}.$$

The column calculation is

$$Ax = 2 \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} - \begin{bmatrix} 3 \\ -1 \\ 4 \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \\ -4 \end{bmatrix}.$$

The row calculation is

$$Ax = \begin{bmatrix} \begin{bmatrix} 1 & 3 \end{bmatrix} x \\ \begin{bmatrix} 2 & -1 \end{bmatrix} x \\ \begin{bmatrix} 0 & 4 \end{bmatrix} x \end{bmatrix} = \begin{bmatrix} -1 \\ 5 \\ -4 \end{bmatrix}.$$

All three calculations produce the same vector.

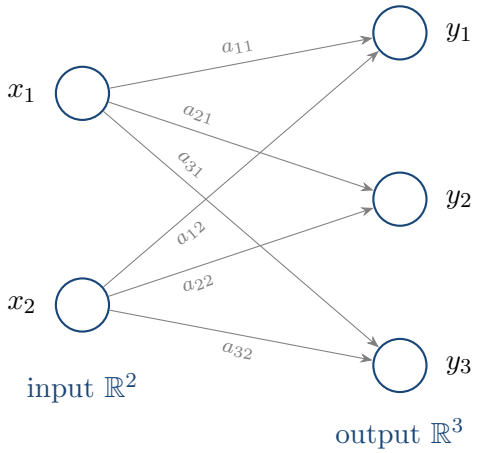
The row and column pictures will be used repeatedly for neural-network layers: rows analyze an input into scalar coordinates, whereas columns synthesize an output as a linear combination.

2.3 A Linear Layer as a Network Diagram

A linear map $\mathbb{R}^2 \to \mathbb{R}^3$ can be drawn as a network with two input nodes and three output nodes. Writing $y = Ax$ with

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \in \mathbb{R}^{3\times 2},$$

each output coordinate is $y_i = a_{i1}x_1 + a_{i2}x_2$. Every input node is joined to every output node by an edge, and the edge from input j to output i carries the weight a_{ij}; the first index names the output and the second names the input.



In this picture the entries of a single *row* of A are the weights on the edges that *arrive* at one output node: row $i = (a_{i1}, a_{i2})$ collects the two edges entering y_i, and y_i is their weighted sum, the *fan-in* of that node. The entries of a single *column* of A are the weights on the edges that *leave* one input node: column $j = (a_{1j}, a_{2j}, a_{3j})^{\mathsf{T}}$ collects the three edges emanating from x_j, the *fan-out* of that node.

Two questions fix the picture. Which edges correspond to the second row of A? They are the two edges entering y_2, namely those labeled a_{21} and a_{22}. Which edges correspond to the first column of A? They are the three edges leaving x_1, namely those labeled a_{11}, a_{21}, and a_{31}.

2.4 Linearity

Definition 2.2. *A function $T : \mathbb{R}^n \to \mathbb{R}^m$ is* linear *if for all $x, y \in \mathbb{R}^n$ and all $c \in \mathbb{R}$,*

$$T(x + y) = T(x) + T(y), \qquad T(cx) = cT(x).$$

Matrix-vector multiplication is linear:

$$A(x + y) = Ax + Ay, \qquad A(cx) = c(Ax).$$

Conversely, every linear map between finite-dimensional Euclidean spaces is matrix multiplication by a unique matrix.

Theorem 2.3 (Matrices represent linear maps)**.** *Let $T : \mathbb{R}^n \to \mathbb{R}^m$ be linear. Then there is a unique matrix $A \in \mathbb{R}^{m \times n}$ such that*

$$T(x) = Ax \quad \text{for all } x \in \mathbb{R}^n.$$

The columns of A are $T(e_1), \dots, T(e_n)$.

Proof. Every $x \in \mathbb{R}^n$ can be written as

$$x = x_1 e_1 + \cdots + x_n e_n.$$

By linearity,

$$T(x) = x_1 T(e_1) + \cdots + x_n T(e_n).$$

If A is the matrix with columns $T(e_1), \dots, T(e_n)$, this last expression is exactly Ax. The matrix is unique because its jth column must equal $Ae_j = T(e_j)$. □

2.5 Rank, Column Space, and Null Space

The column space of $A \in \mathbb{R}^{m \times n}$ is

$$\mathrm{Col}(A) = \{Ax : x \in \mathbb{R}^n\} = \mathrm{span}\{a_1, \dots, a_n\},$$

where $a_1, \dots, a_n$ are the columns of A. It is the set of all possible outputs of the linear map $x \mapsto Ax$.

The null space of A is

$$\mathrm{Null}(A) = \{x \in \mathbb{R}^n : Ax = \mathbf{0}\}.$$

It is the set of input directions that the matrix collapses to zero.

The rank of A, written $\mathrm{rank}(A)$, is the dimension of the column space; equivalently, it is the number of linearly independent output directions available to the transformation. A map of small rank provides a *linear compression*, since its range has small dimension.

Example 2.4 (A rank-one matrix)**.** *Let*

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix}.$$

The second column is twice the first, so the column space is the line

$$\operatorname{span}\left\{ \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \right\}.$$

Therefore $\operatorname{rank}(A) = 1$*. The matrix maps* $\mathbb{R}^2$ *into this line in* $\mathbb{R}^3$*. Its null space is the line*

$$\operatorname{Null}(A) = \operatorname{span}\left\{ \begin{bmatrix} -2 \\ 1 \end{bmatrix} \right\},$$

since $A(-2,1)^{\mathsf{T}} = \mathbf{0}$*.*

2.6 Affine Maps and Bias

An *affine map* is a map of the form

$$x \mapsto Ax + b,$$

where $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. The vector b is called a *bias* or *intercept*. The map is not linear unless $b = \mathbf{0}$, because it does not send $\mathbf{0}$ to $\mathbf{0}$.

Affine maps occur in neural networks and in classification. In particular, a purely linear boundary $w^{\mathsf{T}}x = 0$ passes through the origin, whereas an affine boundary $w^{\mathsf{T}}x + b = 0$ need not do so.

An affine map may be represented as a linear map in one higher dimension:

$$Ax + b = \begin{bmatrix} A & b \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix}.$$

Thus the bias is the coefficient associated with a constant coordinate.

2.7 Matrix Multiplication as Composition

Let $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$. Their product is the matrix $AB \in \mathbb{R}^{m \times p}$ defined by

$$(AB)_{ik} = \sum_{j=1}^{n} a_{ij} b_{jk}.$$

Equivalently, the product represents composition in the order

$$AB \text{ means first apply } B, \text{ then apply } A.$$

Indeed, if $x \in \mathbb{R}^p$, then

$$(AB)x = A(Bx).$$

Thus matrix multiplication is composition of linear maps.

Example 2.5 (Composition)**.** *Let*

$$B = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}.$$

The matrix B stretches the second coordinate by a factor of 2. The matrix A then shears by adding the second coordinate to the first. Their composition is

$$AB = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}.$$

For $x = (x_1, x_2)^\mathsf{T}$,

$$ABx = \begin{bmatrix} x_1 + 2x_2 \\ 2x_2 \end{bmatrix}.$$

2.8 Binary Linear Classification

A *binary classifier* is a function that assigns one of two labels to each input vector. A *binary linear classifier* is determined by an affine score

$$s(x) = w^\mathsf{T} x + b,$$

where $w \in \mathbb{R}^d$ is nonzero and $b \in \mathbb{R}$. Define the sign function by

$$\operatorname{sign}(t) = \begin{cases} +1, & t > 0, \\ 0, & t = 0, \\ -1, & t < 0. \end{cases}$$

The associated classifier is

$$\widehat{y}(x) = \operatorname{sign}(w^\mathsf{T} x + b).$$

Its *decision boundary*, the set on which the score vanishes, is the hyperplane

$$w^\mathsf{T} x + b = 0.$$

On this boundary the displayed classifier has value zero; a classifier with only two output labels must adopt a separate convention there. The vector w is normal to the hyperplane. Among unit directions, the directional change in the score is maximal in the direction $w/\|w\|$ and minimal in the opposite direction.

Example 2.6 (A two-dimensional classifier)**.** *Let*

$$w = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \qquad b = -1.$$

The decision boundary is

$$2x_1 - x_2 - 1 = 0, \qquad \textit{or equivalently} \qquad x_2 = 2x_1 - 1.$$

For $x = (2, 1)^\mathsf{T}$, the score is

$$2 \cdot 2 - 1 - 1 = 2,$$

so the classifier predicts the positive class. For $x = (0, 1)^\mathsf{T}$, the score is

$$0 - 1 - 1 = -2,$$

so the classifier predicts the negative class.

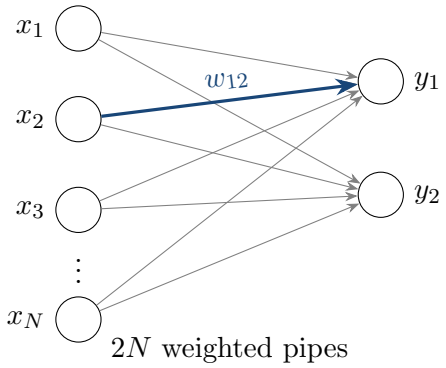
2.9 Pipe Diagrams and the Three Pictures

Consider a linear layer with N input coordinates and two output coordinates. Let

$$W = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1N} \\ w_{21} & w_{22} & \cdots & w_{2N} \end{bmatrix} \in \mathbb{R}^{2\times N}, \qquad \mathbf{y} = W\mathbf{x}.$$

This is the output-by-input convention used throughout these notes: the matrix multiplies a column vector. We use the letter W here to distinguish this matrix from the matrices A and B in the composition below. Writing the product coordinate by coordinate gives

$$\begin{cases} y_1 = w_{11}x_1 + \cdots + w_{1N}x_N, \\ y_2 = w_{21}x_1 + \cdots + w_{2N}x_N. \end{cases}$$



The entrywise picture. The entry w_{ij} is the weight on the pipe from source node j to destination node i. Thus the first index names the output and the second names the input.

The column picture. If $\mathbf{c}_j = (w_{1j}, w_{2j})^{\mathsf{T}}$ is column j, then

$$W\mathbf{x} = x_1\mathbf{c}_1 + \cdots + x_N\mathbf{c}_N.$$

Geometrically, x_j scales $\mathbf{c}_j$ and the layer adds the scaled vectors. Hence

$$\mathrm{Im}(W) = \mathrm{span}\{\mathbf{c}_1, \ldots, \mathbf{c}_N\}.$$

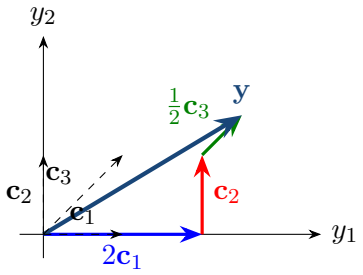
Thus $\mathrm{Im}(W) = \mathrm{Col}(W)$; the two names will be used interchangeably. In $\mathbb{R}^2$ there are three possibilities: the image is $\{0\}$ if all columns vanish; a line if the nonzero columns are parallel; and all of $\mathbb{R}^2$ if at least two columns are independent. The last condition is $\mathrm{rank}(W) = 2$.

For a concrete $N = 3$ sketch, take

$$\mathbf{c}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad \mathbf{c}_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad \mathbf{c}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad (x_1, x_2, x_3) = (2, 1, 1/2).$$

Then the successive vector sum is

$$\mathbf{y} = 2\mathbf{c}_1 + \mathbf{c}_2 + \tfrac{1}{2}\mathbf{c}_3 = \begin{bmatrix} 5/2 \\ 3/2 \end{bmatrix}.$$



The positivity of these coefficients belongs only to this example. The image uses all real coefficients, rather than only nonnegative coefficients. If a bias is added, $\mathbf{y} = W\mathbf{x} + \mathbf{b}$, the attainable set is translated from $\mathrm{Im}(W)$ to the affine set $\mathbf{b} + \mathrm{Im}(W)$.

The row picture. If $r_i^\mathsf{T} = (w_{i1}, \ldots, w_{iN})$ is row i of W, then

$$y_i = r_i^\mathsf{T}\mathbf{x} = \sum_{j=1}^{N} w_{ij}x_j.$$

Each row is a single linear functional on the input, of exactly the kind that formed the score of a binary linear classifier above. Its level set $\{\mathbf{x} : r_i^\mathsf{T}\mathbf{x} = c\}$ is a hyperplane, and the two open half-spaces on either side are the half-space features recorded by the sign of $r_i^\mathsf{T}\mathbf{x}$. The collection of rows of W is therefore a collection of such half-space features, one per row. In the pipe diagram, row i collects all pipes arriving at output i—its *fan-in*—whereas column j collects all pipes leaving input j—its *fan-out*.

2.10 Linear Separability and Feature Choice

A binary labeled data set

$$(x^{(1)}, y^{(1)}), \ldots, (x^{(N)}, y^{(N)}), \qquad y^{(i)} \in \{-1, +1\},$$

is *linearly separable* if there exist $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ such that

$$y^{(i)}(w^\mathsf{T}x^{(i)} + b) > 0 \quad \text{for all } i.$$

This condition says that all positive examples lie strictly on one side of a hyperplane and all negative examples lie strictly on the other.

Linear separability depends on the representation. A data set may not be separable in the original coordinates but may become separable after a feature map

$$\phi : \mathbb{R}^d \to \mathbb{R}^p.$$

In that case a classifier that is linear in $\phi(x)$ may be nonlinear as a function of the original input x.

Example 2.7 (A nonlinear feature makes a linear classifier possible)**.** *The points $x = -2$ and $x = 2$ are labeled positive, while $x = 0$ is labeled negative. These three points are not separable on the real line by a threshold that puts both extremes on one side and the middle on the other. But with the feature map*

$$\phi(x) = x^2,$$

the positive examples have feature value 4 *and the negative example has feature value* 0*. The rule*

$$\phi(x) - 2 > 0$$

separates them. This example exhibits the role of a feature map. In a neural network, the parameters of hidden layers determine such features from the data rather than requiring them to be prescribed in advance.

3 Nonlinearities and Feedforward Neural Networks

3.1 Neural Network Layers

A *densely connected neural-network layer* is a map in which every output coordinate is permitted to depend on every input coordinate. Acting on a single input vector $x \in \mathbb{R}^d$, it has the form

$$z = Ax + b, \qquad h = \sigma(z).$$

Here $A \in \mathbb{R}^{k \times d}$, $b \in \mathbb{R}^k$, and $\sigma : \mathbb{R}^k \to \mathbb{R}^k$ is obtained by applying a scalar function to each coordinate. This scalar function is the *activation function*. Common choices are the rectified linear unit, the hyperbolic tangent, and the logistic function:

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

A *multilayer perceptron* is a composition of such layers. The same map is applied to each input vector individually.

Example 3.1 (A small ReLU layer)**.** *Let*

$$x = \begin{bmatrix} 1 \\ -2 \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & 2 \\ -1 & 1 \\ 2 & 0 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix}.$$

Then

$$Ax + b = \begin{bmatrix} -3 \\ -3 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ -1 \end{bmatrix} = \begin{bmatrix} -3 \\ -2 \\ 1 \end{bmatrix}.$$

After applying ReLU,

$$h = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}.$$

The layer has converted a two-dimensional input into a three-dimensional representation.

3.2 The Role of Nonlinear Activation Functions

A *feedforward neural network* is a finite composition of affine maps and coordinatewise activation functions. With several densely connected layers, it has the form

$$h_1 = \sigma_1(A_1 x + b_1),$$

$$h_2 = \sigma_2(A_2 h_1 + b_2),$$

and so on. The matrices and biases are parameters; the activation functions are fixed as part of the chosen family.

If the nonlinearities are removed, depth alone does not create a richer class of functions.

Proposition 3.2 (Compositions of affine maps are affine)**.** *Let*

$$f(x) = Ax + a, \qquad g(x) = Bx + b.$$

Then $f \circ g$ is affine:

$$(f \circ g)(x) = ABx + (Ab + a).$$

Consequently, a network made only of affine layers is equivalent to a single affine layer.

Proof. Compute directly:

$$f(g(x)) = A(Bx + b) + a = ABx + Ab + a.$$

The result is again affine. □

Thus a non-affine activation is necessary if the resulting family is to contain functions that are not affine. For example, ReLU divides the input space into regions on each of which the network is affine; subsequent affine maps may separate representations that were not linearly separable before the activation was applied.

3.3 Rows, Columns, and Learned Features

There are three descriptions of a neural-network layer. The first is the *row picture*. Write an affine layer as

$$a(x) = Ax + b, \qquad A \in \mathbb{R}^{k \times d}, \qquad b \in \mathbb{R}^k.$$

If the rows of A are $w_1^\mathsf{T}, \ldots, w_k^\mathsf{T}$, then

$$a_j(x) = w_j^\mathsf{T} x + b_j.$$

Thus each row measures one affine score. The zero set

$$\{x \in \mathbb{R}^d : w_j^\mathsf{T} x + b_j = 0\}$$

is a hyperplane with normal vector w_j, provided $w_j \neq \mathbf{0}$. In two dimensions, each nonzero hidden coordinate therefore determines a line and the two half-planes bounded by it. The logistic and tanh functions give smooth functions of the corresponding affine score, whereas ReLU gives a function that vanishes on one closed half-plane and is affine on the other.

The second is the *column picture*. Let $U \in \mathbb{R}^{m \times k}$ and $h \in \mathbb{R}^k$. If the columns of U are $u^{(1)}, \ldots, u^{(k)} \in \mathbb{R}^m$, then

$$Uh = \sum_{j=1}^{k} h_j u^{(j)}.$$

In the absence of non linear activation, the output is a linear combination of the columns of U . The coefficients are the coordinates of h, and every possible output lies in the column space of U. In a neural network, h is the activation vector produced by the preceding layer.

The mixed picture combines the two readings. A two-layer network with input in $\mathbb{R}^d$ and output in $\mathbb{R}^m$ can be written as

$$f(x) = U\sigma(Ax + b) + c, \qquad f : \mathbb{R}^d \to \mathbb{R}^m,$$

where $A \in \mathbb{R}^{k\times d}$, $b \in \mathbb{R}^k$, $U \in \mathbb{R}^{m\times k}$, $c \in \mathbb{R}^m$, and

$$h(x) = \sigma(Ax + b) \in \mathbb{R}^k$$

is the vector of k hidden features. Reading A by rows, the first layer produces affine scores and hidden features:

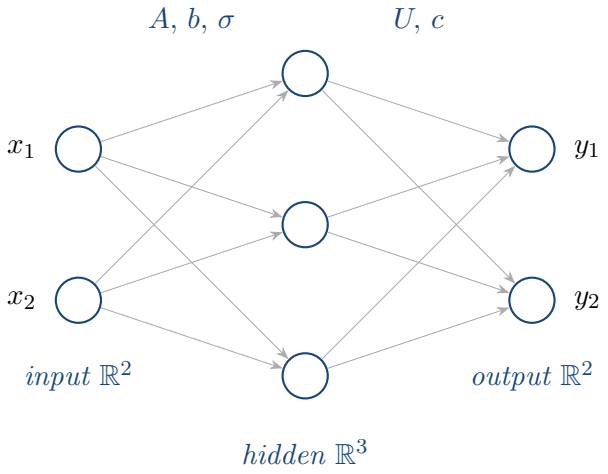
$$h_j(x) = \sigma(w_j^{\mathsf{T}} x + b_j).$$

Reading U by columns, the second layer combines these features into an output:

$$f(x) = \sum_{j=1}^{k} h_j(x) u^{(j)} + c.$$

Accordingly, the rows specify the scalar features, whereas the columns specify the contribution of each feature to the output.

Example 3.3 (Row features and column synthesis)**.** *With input dimension* 2*, hidden dimension* 3*, and output dimension* 2*, the two-layer map* $x \mapsto U\sigma(Ax + b) + c$ *has the network diagram*



Let

$$A = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ -2 \end{bmatrix}, \qquad x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}.$$

The first-layer pre-activation is

$$Ax + b = \begin{bmatrix} \frac{1}{2} \\ -\frac{3}{2} \\ 1 \end{bmatrix},$$

and therefore

$$h = \mathrm{ReLU}(Ax + b) = \begin{bmatrix} \frac{1}{2} \\ 0 \\ 1 \end{bmatrix}.$$

Let

$$U = \begin{bmatrix} 2 & -1 & \frac{1}{2} \\ -1 & 2 & 1 \end{bmatrix},$$

and take the output bias to be $c = \mathbf{0}$*. Writing the columns of* U *as*

$$u^{(1)} = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad u^{(2)} = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \quad u^{(3)} = \begin{bmatrix} \frac{1}{2} \\ 1 \end{bmatrix},$$

the output is

$$Uh = \frac{1}{2}u^{(1)} + 0u^{(2)} + u^{(3)} = \begin{bmatrix} \frac{3}{2} \\ \frac{1}{2} \end{bmatrix}.$$

The rows of A *determine which feature coordinates are nonzero; the columns of* U *determine their linear contributions to the output.*

A deep neural network repeats this construction through many layers. It is therefore a parametrized composition of elementary linear-algebraic operations and fixed nonlinear functions.

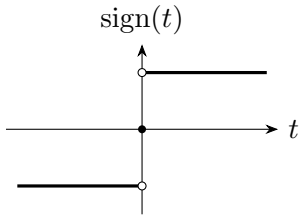
3.4 Lines, Half-Spaces, and Weighted Sign Features

We now study the sign patterns produced by three affine functionals and their combination in a network with one hidden layer.

The activation is the sign function already used in binary classification:

$$\operatorname{sign}(t) = \begin{cases} +1, & t > 0, \\ 0, & t = 0, \\ -1, & t < 0. \end{cases}$$

Its graph has two horizontal open rays at heights ± 1 and the single filled point $(0, 0)$ at the boundary.



3.4.1 Three Lines in General Position

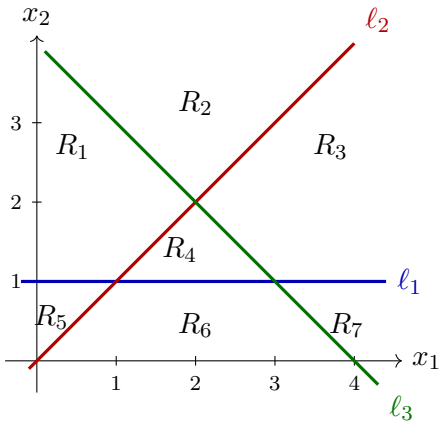
Let $\mathbf{x} = (x_1, x_2)^{\mathsf{T}}$ and consider

$$\mathbf{u}_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ c_1 = -1, \qquad \mathbf{u}_2 = \begin{bmatrix} -1 \\ 1 \end{bmatrix}, \ c_2 = 0, \qquad \mathbf{u}_3 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ c_3 = -4.$$

The equations $\mathbf{u}_i^{\mathsf{T}}\mathbf{x} + c_i = 0$ are

$$\ell_1 : x_2 = 1, \qquad \ell_2 : x_2 = x_1, \qquad \ell_3 : x_1 + x_2 = 4.$$

No two of these lines are parallel and the three do not meet at a common point; the arrangement is in *general position*. Adding the lines one at a time divides the plane into $1 + 1 + 2 + 3 = 7$ regions, and it is these regions—rather than the pairwise intersection points—that the sign features will label.



For each i, define the open half-space and its sign classifier by

$$A_i = \{\mathbf{x} : \mathbf{u}_i^\mathsf{T}\mathbf{x} + c_i > 0\}, \qquad f_i(\mathbf{x}) = \operatorname{sign}(\mathbf{u}_i^\mathsf{T}\mathbf{x} + c_i).$$

Then $f_i = 1$ on A_i, $f_i = 0$ on the boundary ℓ_i, and $f_i = -1$ on the opposite open half-space. Testing one point strictly inside each cell gives

cell	one possible test point	(f_1, f_2, f_3)
R_1	$(1/2, 5/2)$	$(+,+,-)$
R_2	$(2, 3)$	$(+,+,+)$
R_3	$(7/2, 5/2)$	$(+,-,+)$
R_4	$(2, 3/2)$	$(+,-,-)$
R_5	$(1/4, 1/2)$	$(-,+,-)$
R_6	$(2, 1/2)$	$(-,-,-)$
R_7	$(4, 1/2)$	$(-,-,+)$

Three signs allow eight formal patterns, but only seven are realized by this arrangement; $(-,+,+)$ is impossible. Thus a sign vector labels a region only when the corresponding pattern is geometrically realizable. Indeed, $f_1 = -1$ and $f_2 = +1$ imply $x_2 < 1$ and $x_1 < x_2$, hence $x_1 + x_2 < 2$, contradicting $f_3 = +1$, which requires $x_1 + x_2 > 4$.

Let us now add another layer. Given f_i's defined above, let $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), f_2(\mathbf{x}), f_3(\mathbf{x}))^\mathsf{T}$ and define the aggregate output by

$$F(\mathbf{x}) = \mathbf{w}^\mathsf{T}\mathbf{f}(\mathbf{x}).$$

For $\mathbf{w} = (0, 0, 1)^\mathsf{T}$,

$$F(\mathbf{x}) = f_3(\mathbf{x}) = \operatorname{sign}(x_1 + x_2 - 4).$$

Its value is $+1$ precisely in $A_3 = \{x_1 + x_2 > 4\}$, the union of R_2 and R_3 in the displayed arrangement. It is -1 on the other five open regions and 0 on ℓ_3. Thus F is a weighted linear combination of the three sign functions: the first-layer rows determine half-space features, and the output weights combine them. This is a one-hidden-layer network with sign activations and a linear output. In the terminology of *boosting*, a method that combines component classifiers, the half-space rules are base classifiers and their weighted combination is an ensemble.

For four sign features, suppose $h_i(\mathbf{x}) \in \{-1, 1\}$, put $h(\mathbf{x}) = (h_1(\mathbf{x}), \ldots, h_4(\mathbf{x}))^\mathsf{T}$, and let a desired cell have target pattern $\mathbf{w} = (1, -1, 1, -1)^\mathsf{T}$. Then

$$\mathbf{w}^\mathsf{T}h(\mathbf{x}) \le 4,$$

with equality exactly when $h(\mathbf{x}) = \mathbf{w}$. For a set E, write $\mathbf{1}_E$ for its indicator function, equal to 1 on E and 0 outside E. The preceding score is not itself a $\{0,1\}$-valued indicator. Away from boundaries, the desired indicator is, for example,

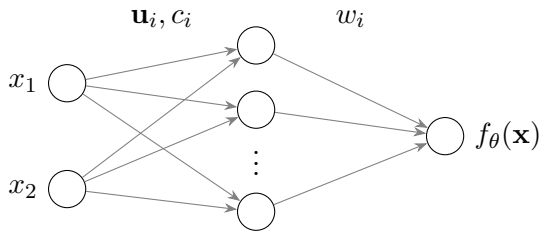
$$\mathbf{1}_{\{\mathbf{w}^\mathsf{T} h > 3\}}.$$

Here $A_i' = \{\mathbf{x} : \mathbf{u}_i^\mathsf{T}\mathbf{x} + c_i < 0\}$ denotes the open half-space opposite to A_i. The formula gives the strict region $R = A_1 \cap A_2' \cap A_3 \cap A_4'$ and excludes boundary patterns containing zeros.

More generally, for N sign features let $h(\mathbf{x}), \mathbf{w} \in \{-1,1\}^N$. Distinct sign vectors have dot product at most $N-2$, while $\mathbf{w}^\mathsf{T} h(\mathbf{x}) = N$ holds exactly when $h(\mathbf{x}) = \mathbf{w}$. Equality with the maximum score N therefore tests for the target pattern.

For the network $\mathbb{R}^2 \to \mathbb{R}^N \to \mathbb{R}$, the parameters are

$$\theta = (\mathbf{u}_1, \ldots, \mathbf{u}_N, c_1, \ldots, c_N, \mathbf{w}), \qquad \dim \Theta = 2N + N + N = 4N.$$



Besides sign, one may use the Heaviside step function $H(t) = 1$ for $t > 0$ and $H(t) = 0$ for $t < 0$, with a separately specified value at zero, or a smooth approximation such as the logistic function. The value assigned on the boundary depends on this choice.

3.5 Parameters Versus Functions

The parameter-to-function map

$$\theta \in \Theta \longmapsto f_\theta$$

need not be injective; that is, distinct parameters may determine the same function. Define an equivalence relation by

$$\theta \sim \theta' \quad \Longleftrightarrow \quad f_\theta = f_{\theta'}.$$

For the homogeneous sign classifier

$$f_\theta(x) = \operatorname{sign}(\theta^\mathsf{T} x), \qquad \theta \in \mathbb{R}^2 \setminus \{0\},$$

positive rescaling does not change the function: $f_{r\theta} = f_\theta$ for $r > 0$. Every positive ray meets the unit circle once, so the set of equivalence classes, called the *quotient* of the parameter space, is identified with $S^1 = \{u \in \mathbb{R}^2 : \|u\| = 1\}$. Negative rescaling is not identified because it reverses the signs.

The equivalence class of θ is denoted by $[\theta]$. Let $\Omega \subseteq \mathbb{R}^2$ be a region of finite area (for example a cube) for which the following integral is defined. One comparison of the restrictions of the functions to Ω is

$$d([\theta],[\theta']) = \int_\Omega |f_\theta(x) - f_{\theta'}(x)| \, \mathrm{d}x.$$

The displayed function is a *pseudometric*: it is nonnegative and symmetric, satisfies the triangle inequality, and vanishes when its two arguments agree, but it may also vanish on two distinct classes whose restrictions to Ω differ only on a subset of Ω having area zero. The restrictions are then said to

be equal *almost everywhere on* Ω. Identifying restrictions that are equal almost everywhere on Ω gives a metric. No general conclusion that this quotient is a smooth manifold—a space locally modeled on Euclidean space by smoothly compatible coordinates—follows from the construction; that property requires a separate argument.

3.6 Exercises

These exercises consolidate the linear-algebraic ideas developed above.

Exercise 3.4 (Linear combinations)**.** *Let*

$$v_1 = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix}, \qquad v_2 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}, \qquad v_3 = \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix}.$$

Show that v_3 lies in $\operatorname{span}\{v_1, v_2\}$. Is the representation unique?

Exercise 3.5 (Inner products and angles)**.** *For*

$$x = \begin{bmatrix} 1 \\ 2 \\ 2 \end{bmatrix}, \qquad y = \begin{bmatrix} 2 \\ 0 \\ 1 \end{bmatrix},$$

compute $\langle x, y \rangle$, $\|x\|$, $\|y\|$, and the cosine of the angle between x and y.

Exercise 3.6 (Projection)**.** *Let $u = (1, 2)^{\mathsf{T}}$ and $x = (4, 1)^{\mathsf{T}}$. Compute $\operatorname{proj}_u(x)$ and verify that $x - \operatorname{proj}_u(x)$ is orthogonal to u.*

Exercise 3.7 (Matrix-vector multiplication)**.** *Let*

$$A = \begin{bmatrix} 2 & 0 & 1 \\ -1 & 3 & 2 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 \\ -2 \\ 4 \end{bmatrix}.$$

Compute Ax using both the row view and the column view.

Exercise 3.8 (Rank and null space)**.** *For the matrix*

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 3 & 6 \end{bmatrix},$$

find the column space, the rank, and a nonzero vector in the null space.

Exercise 3.9 (A linear map from its action on basis vectors)**.** *Suppose $T : \mathbb{R}^3 \to \mathbb{R}^2$ is linear and*

$$T(e_1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad T(e_2) = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \quad T(e_3) = \begin{bmatrix} 0 \\ 3 \end{bmatrix}.$$

Write the matrix A such that $T(x) = Ax$.

Exercise 3.10 (Matrix multiplication)**.** *Let*

$$A = \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix}, \qquad B = \begin{bmatrix} 3 & 1 \\ 2 & 0 \end{bmatrix}.$$

Compute AB and BA. Are they equal?

Exercise 3.11 (A binary classifier)**.** *Consider the classifier*

$$\widehat{y}(x) = \operatorname{sign}(3x_1 - 2x_2 + 1).$$

Find the decision boundary. Classify the points $(0,0)^\mathsf{T}$*,* $(1,3)^\mathsf{T}$*, and* $(-1,0)^\mathsf{T}$*.*

Exercise 3.12 (Affine layers collapse without nonlinearities)**.** *Let*

$$f_1(x) = A_1x + b_1, \qquad f_2(h) = A_2h + b_2, \qquad f_3(u) = A_3u + b_3.$$

Find a matrix A *and vector* b *such that*

$$f_3(f_2(f_1(x))) = Ax + b.$$

Exercise 3.13 (A ReLU layer)**.** *Let*

$$A = \begin{bmatrix} 1 & 3 \\ -2 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, \qquad x = \begin{bmatrix} -1 \\ 2 \end{bmatrix}.$$

Compute

$$z = Ax + b \quad \textit{and} \quad h = \operatorname{ReLU}(z).$$

Exercise 3.14 (Row scores and column synthesis)**.** *Let*

$$A = \begin{bmatrix} 1 & -1 \\ -1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad b = \begin{bmatrix} -\frac{1}{2} \\ -\frac{1}{2} \\ -2 \end{bmatrix}, \qquad U = \begin{bmatrix} 2 & -1 & \frac{1}{2} \\ -1 & 2 & 1 \end{bmatrix}.$$

Take the output bias to be zero. For $x = (1,3)^\mathsf{T}$*, compute*

$$h = \operatorname{ReLU}(Ax + b) \qquad \textit{and} \qquad y = Uh.$$

Then write y *as a linear combination of the columns of* U*.*

4 The Singular Value Decomposition and Linear Autoencoders

From this section onward, vectors are again denoted by ordinary italic letters; the boldface used in the network diagrams above carries no mathematical distinction.

4.1 The Mixed Picture: Encoder and Decoder

Let

$$A : \mathbb{R}^n \to \mathbb{R}^k, \qquad B : \mathbb{R}^k \to \mathbb{R}^n, \qquad k < n,$$

with $A \in \mathbb{R}^{k\times n}$ and $B \in \mathbb{R}^{n\times k}$. A network with no activation computes $f(\mathbf{x}) = BA\mathbf{x}$. The map A is the *encoder*, which sends the input to its code, and B is the *decoder*, which sends the code back to the input space. Write row j of A as a_j^T and column j of B as b_j. The row picture first gives

$$z = A\mathbf{x} = \begin{bmatrix} a_1^\mathsf{T}\mathbf{x} \\ \vdots \\ a_k^\mathsf{T}\mathbf{x} \end{bmatrix}.$$

The vector $z \in \mathbb{R}^k$ is the *latent vector*, or *code*. Its k coordinates are the displayed dot products. They are not orthogonal projection coefficients in input space unless the rows of A have the required orthonormality. The column picture then gives

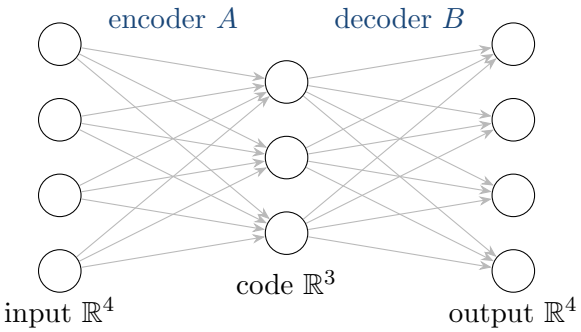
$$Bz = z_1 b_1 + \cdots + z_k b_k = \sum_{j=1}^{k} (a_j^{\mathsf{T}} \mathbf{x}) b_j.$$

Consequently

$$BA = \sum_{j=1}^{k} b_j a_j^{\mathsf{T}}.$$

This is the mixed picture: rows analyze the input; columns synthesize the output.

For the concrete architecture $n = 4$ and $k = 3$, every one of the four input nodes is connected to every one of the three hidden nodes through A, and every hidden node is connected to every output node through B. Each layer therefore has 12 edges. No activation is inserted, so the two layers compose to the single matrix BA.



Because every output lies in $\operatorname{span}\{b_1, \ldots, b_k\}$,

$$\operatorname{Im}(BA) \subseteq \operatorname{Im}(B), \qquad \operatorname{rank}(BA) \leq \min\{\operatorname{rank}(A), \operatorname{rank}(B)\} \leq k.$$

The maximum rank is k, attained when A has full row rank k and B has full column rank k. Conversely, every matrix satisfying this rank bound has such a factorization.

Proposition 4.1 (Rank factorizations)**.** *Every $M \in \mathbb{R}^{n \times n}$ with $\operatorname{rank}(M) \leq k$ can be written $M = BA$ with $A \in \mathbb{R}^{k \times n}$ and $B \in \mathbb{R}^{n \times k}$.*

Proof. If $r = \operatorname{rank}(M)$, select a basis $b_1, \ldots, b_r$ for $\operatorname{Col}(M)$ and put those vectors into $B_r \in \mathbb{R}^{n \times r}$. Each column of M has a coordinate vector in this basis; collect these in $A_r \in \mathbb{R}^{r \times n}$. Then $M = B_r A_r$. If $r < k$, add zero columns to B_r and zero rows to A_r. □

4.1.1 Exact Reconstruction Through a Low-Rank Map

For data $x^{(1)}, \ldots, x^{(N)} \in \mathbb{R}^n$, define

$$E(A, B) = \sum_{i=1}^{N} \left\| x^{(i)} - BAx^{(i)} \right\|^2.$$

The condition $E = 0$ means that BA is the identity on every data point. Since $\mathrm{Im}(BA)$ has dimension at most k, this can happen only if the data span a subspace of dimension at most k. Conversely, if their span lies in some k-dimensional subspace S, choose B to span S and choose A so that BA is the identity on S (the orthogonal projector is one choice). Hence zero error is possible if and only if $\dim \operatorname{span}\{x^{(1)}, \ldots, x^{(N)}\} \le k$. If this dimension condition fails, the loss is necessarily positive; the rank bound prevents exact reconstruction.

For approximate reconstruction, select a k-dimensional subspace S, let B have orthonormal columns spanning S, and take $A = B^{\mathsf{T}}$. Then $BA = BB^{\mathsf{T}}$ is the orthogonal projector onto S. The remaining problem is to choose the subspace that gives the smallest total squared distance. For *centered data*, meaning $\sum_{i=1}^{N} x^{(i)} = 0$, minimizing this quantity over all k-dimensional subspaces is called *principal component analysis* (PCA). For data that are not centered, the construction finds the best linear subspace constrained to pass through the origin. With sample mean $\mu = N^{-1} \sum_{i=1}^{N} x^{(i)}$, affine PCA subtracts and later restores it:

$$\widehat{x} = \mu + U_k U_k^{\mathsf{T}} (x - \mu).$$

Here the columns of $U_k \in \mathbb{R}^{n \times k}$ form an orthonormal basis for the selected k-dimensional subspace; the optimal choice is derived below from the singular value decomposition.

4.2 Orthogonal Maps and High-Dimensional Geometry

An orthonormal basis $q_1, \ldots, q_n$ satisfies $q_i^{\mathsf{T}} q_j = \delta_{ij}$, where the Kronecker delta δ_{ij} is 1 when $i = j$ and 0 otherwise. The matrix $Q = [q_1 \ \cdots \ q_n]$ obeys

$$Q^{\mathsf{T}} Q = Q Q^{\mathsf{T}} = I, \qquad Q^{-1} = Q^{\mathsf{T}}.$$

Such a matrix is *orthogonal*. Its column picture is a change from the standard orthonormal basis to another orthonormal basis:

$$Qx = x_1 q_1 + \cdots + x_n q_n.$$

The determinant $\det Q$ is the signed factor by which Q scales n-dimensional volume. An orthogonal map preserves inner products, lengths, angles, and volume magnitude:

$$\langle Qx, Qy \rangle = \langle x, y \rangle, \qquad \|Qx\| = \|x\|, \qquad |\det Q| = 1.$$

The sign of the determinant defines orientation: orthogonal maps with $\det Q = 1$ preserve orientation, whereas those with $\det Q = -1$ reverse it. Rotations and reflections are the respective elementary examples.

Several elementary facts become more consequential in high dimension. A single direction spans only a one-dimensional subspace; k independent directions span at most k dimensions no matter how large the ambient space is. Orthogonal projection onto their span is therefore a substantial compression when $k \ll n$. The singular value decomposition identifies the directions that account for the largest squared norms.

For a probabilistic example, let every coordinate of $x, y \in \mathbb{R}^n$ be chosen independently and uniformly from $\{-1/\sqrt{n}, 1/\sqrt{n}\}$. Thus all 2^{2n} ordered pairs (x, y) are equally likely. For any quantity $g(x, y)$, $\mathbb{E}g$ denotes its average over these pairs. Then $\|x\| = \|y\| = 1$ and

$$\mathbb{E}\ \langle x, y \rangle = 0, \qquad \mathbb{E}\ \langle x, y \rangle^2 = \frac{1}{n},$$

so the root-mean-square cosine similarity, $(\mathbb{E}\ \langle x, y \rangle^2)^{1/2}$, is $n^{-1/2}$. In high-dimensional spaces, this estimate motivates the use of directions, projections, and low-dimensional subspaces in place of coordinate drawings.

4.3 The Singular Value Decomposition

Theorem 4.2 (Singular value decomposition)**.** *For every $M \in \mathbb{R}^{m\times n}$ there are orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ such that*

$$M = U\Sigma V^{\mathsf{T}},$$

where $\Sigma \in \mathbb{R}^{m\times n}$ is diagonal in the rectangular sense. Its nonzero diagonal entries may be ordered

$$\sigma_1 \geq \cdots \geq \sigma_r > 0, \qquad r = \operatorname{rank}(M).$$

The diagonal entries σ_i are the *singular values*. The columns v_i of V are the *right singular vectors* and form an orthonormal basis of the input space; the columns u_i of U are the *left singular vectors* and form an orthonormal basis of the output space. For the nonzero singular directions, the defining relations are

$$Mv_i = \sigma_i u_i, \qquad M^{\mathsf{T}} u_i = \sigma_i v_i \quad (1 \leq i \leq r).$$

Consequently

$$Mx = \sum_{i=1}^{r} \sigma_i (v_i^{\mathsf{T}} x) u_i, \qquad M = \sum_{i=1}^{r} \sigma_i u_i v_i^{\mathsf{T}}.$$

Thus the mixed description has orthonormal analysis and synthesis directions.

The three stages have a geometric reading:

$$x \overset{V^{\mathsf{T}}}{\longmapsto} V^{\mathsf{T}}x \overset{\Sigma}{\longmapsto} \Sigma V^{\mathsf{T}}x \overset{U}{\longmapsto} U\Sigma V^{\mathsf{T}}x.$$

The first stage reads right-singular coordinates, the second scales coordinate i by σ_i, and the third constructs the output in the left-singular basis.

Example 4.3 (The rectangular diagonal picture)**.** *For*

$$\Sigma = \begin{bmatrix} 3 & 0 \\ 0 & 2 \\ 0 & 0 \end{bmatrix} : \mathbb{R}^2 \to \mathbb{R}^3,$$

the unit circle in the input plane is sent to an ellipse with semiaxes 3 and 2 in the e_1e_2-plane of $\mathbb{R}^3$. Composition on the right by V^{T} changes the orthonormal input coordinates, and composition on the left by U changes the orthonormal output coordinates.

The *reduced SVD* discards the columns corresponding to zero singular values:

$$M = U_r \Sigma_r V_r^{\mathsf{T}},$$

with shapes $U_r : m \times r$, $\Sigma_r : r \times r$, and $V_r : n \times r$. In particular, the displayed dimensions distinguish vectors in the input and output spaces.

4.3.1 Frobenius Norm and Truncated SVD

For a square matrix $G = (g_{ij})$, its *trace* is $\operatorname{tr}(G) = \sum_i g_{ii}$. For an arbitrary matrix $C = (c_{ij})$, its *Frobenius norm* is defined by

$$\|C\|_F^2 = \sum_{i,j} c_{ij}^2 = \operatorname{tr}(C^{\mathsf{T}}C).$$

In the 2×2 case,

$$C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} \quad \Longrightarrow \quad \|C\|_F^2 = c_{11}^2 + c_{12}^2 + c_{21}^2 + c_{22}^2.$$

Orthogonal multiplication preserves this norm. The rank-k truncation of M is

$$M_k = \sum_{i=1}^{k} \sigma_i u_i v_i^{\mathsf{T}}.$$

The *spectral norm* of a matrix C is

$$\|C\|_2 = \max_{\|x\|=1} \|Cx\|.$$

Theorem 4.4 (Eckart–Young–Mirsky)**.** *For $0 \le k < r = \operatorname{rank}(M)$, among all matrices N with $\operatorname{rank}(N) \le k$, M_k minimizes both the Frobenius and spectral errors. Precisely,*

$$\min_{\operatorname{rank}(N)\le k} \|M - N\|_F = \left(\sum_{i>k} \sigma_i^2 \right)^{1/2}, \qquad \min_{\operatorname{rank}(N)\le k} \|M - N\|_2 = \sigma_{k+1}.$$

Thus directions associated with small singular values contribute little to the stated approximation errors. If singular values tie at the cutoff, the minimizing subspace need not be unique.

4.4 Least Squares from the SVD

Given $M \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$, consider

$$\min_x \|Mx - b\|_2^2.$$

Write $M = U\Sigma V^{\mathsf{T}}$, $z = V^{\mathsf{T}}x$, and $c = U^{\mathsf{T}}b$. Since U preserves length,

$$\|Mx - b\| = \|\Sigma z - c\|.$$

For $i \le r$, the minimizing coordinates are $z_i = c_i/\sigma_i$. Coordinates in $\operatorname{Null}(M)$ do not change the residual; setting them to zero selects the solution of smallest norm. Define Σ^+ by transposing the rectangular shape and replacing every nonzero σ_i with $1/\sigma_i$. The matrix $M^+ = V\Sigma^+U^{\mathsf{T}}$ is the *Moore–Penrose pseudoinverse* of M, and the solution of smallest norm is

$$x_* = M^+ b.$$

The residual $b - Mx_*$ is orthogonal to $\operatorname{Col}(M)$, which is equivalent to the *normal equations* $M^{\mathsf{T}}Mx = M^{\mathsf{T}}b$. When M has full column rank, their solution is

$$x_* = (M^{\mathsf{T}}M)^{-1}M^{\mathsf{T}}b.$$

Only nonzero singular values may be inverted. Very small singular values amplify perturbations in b; the problem is then called *ill-conditioned*. Truncation is one form of *regularization*, meaning a modification of the problem that limits this sensitivity.

4.4.1 Linear Regression as Least Squares

For observations (a_i, b_i) with feature rows a_i^T, assemble the design matrix and response vector

$$M = \begin{bmatrix} a_1^\mathsf{T} \\ \vdots \\ a_N^\mathsf{T} \end{bmatrix}, \qquad b = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix}.$$

A coefficient vector x predicts Mx and leaves residual $b - Mx$; ordinary linear regression is precisely $\min_x \|Mx - b\|^2$. An intercept is included by appending a column of ones to M. The SVD solution above gives the minimum-norm regression coefficients and identifies sensitive feature directions through small singular values.

4.5 Linear Autoencoders and the SVD

Fix $1 \le k < d$. For the truncated-factor notation below, assume $k \le N$. If $N < k$, the data span has dimension at most $N < k$, and the exact reconstruction criterion already applies. Let $x^{(1)}, \dots, x^{(N)} \in \mathbb{R}^d$ be centered data, meaning $\sum_{i=1}^N x^{(i)} = 0$, and store them as columns of

$$X = [x^{(1)} \ \cdots \ x^{(N)}] \in \mathbb{R}^{d \times N}.$$

Here each data point is a column of X; we adopt this column convention because XX^T then acts directly on feature space. An encoder $E \in \mathbb{R}^{k \times d}$ and decoder $D \in \mathbb{R}^{d \times k}$ compute

$$z = Ex, \qquad \widehat{x} = Dz = DEx.$$

A *linear autoencoder* is such a pair of linear maps whose parameters are chosen so that DEx approximates x on the specified data. The letters E and D distinguish the encoder and decoder from the factors used in the singular value decomposition. For all samples at once,

$$\mathcal{L}(E, D) = \sum_{i=1}^{N} \left\| x^{(i)} - DEx^{(i)} \right\|^2 = \|X - DEX\|_F^2 .$$

The product DE has rank at most k, and every rank-at-most-k matrix admits such a factorization. Thus the encoder–decoder factorization imposes precisely a rank constraint on the composite linear map.

4.5.1 Reduction to Orthogonal Projection

Fix the decoder subspace $S = \mathrm{Col}(D)$. For each input, the closest point of S is its orthogonal projection. If $\dim S = r < k$, enlarge S to a k-dimensional space; this cannot increase the best error. We may therefore take $\dim S = k$. Choose an orthonormal basis of S and place its vectors in the columns of D. Then $D^\mathsf{T} D = I_k$, the encoder may be chosen as $E = D^\mathsf{T}$, and

$$\widehat{x} = DD^\mathsf{T} x.$$

The reconstruction and residual are orthogonal, so with $P = DD^\mathsf{T}$,

$$\mathrm{tr}\Big((PX)^\mathsf{T}(X - PX)\Big) = \mathrm{tr}\Big(X^\mathsf{T} P(I - P)X\Big) = 0,$$

because $P^\mathsf{T} = P$ and $P^2 = P$. Therefore

$$\|X\|_F^2 = \left\|DD^\mathsf{T}X\right\|_F^2 + \left\|X - DD^\mathsf{T}X\right\|_F^2.$$

Moreover,

$$\left\|DD^\mathsf{T}X\right\|_F^2 = \left\|D^\mathsf{T}X\right\|_F^2 = \mathrm{tr}(D^\mathsf{T}XX^\mathsf{T}D).$$

Thus minimizing reconstruction error is equivalent to maximizing the total squared length of the codes. Let $X = U\Sigma V^\mathsf{T}$ and define the truncated factors

$$U_k \in \mathbb{R}^{d\times k}, \qquad \Sigma_k \in \mathbb{R}^{k\times k}, \qquad V_k \in \mathbb{R}^{N\times k}.$$

When an index exceeds $\mathrm{rank}(X)$, the corresponding diagonal entry of Σ is understood to be the singular value $\sigma_i = 0$. The columns of U_k are the first k left singular vectors. Since

$$XX^\mathsf{T} = U\Sigma\Sigma^\mathsf{T}U^\mathsf{T},$$

these vectors are eigenvectors of XX^T with eigenvalues σ_i^2: by definition, a nonzero vector u is an eigenvector of a square matrix C with eigenvalue λ if $Cu = \lambda u$. To see optimality directly, define $\alpha_i = \left\|D^\mathsf{T}u_i\right\|^2$. Expanding the columns of D in the orthonormal basis u_i gives

$$\mathrm{tr}(D^\mathsf{T}XX^\mathsf{T}D) = \sum_i \sigma_i^2\alpha_i, \qquad 0 \le \alpha_i \le 1, \quad \sum_i \alpha_i = k.$$

Indeed, $0 \le \alpha_i \le 1$ follows because DD^T is an orthogonal projector, and $\sum_i \alpha_i = \sum_i u_i^\mathsf{T}DD^\mathsf{T}u_i = \mathrm{tr}(DD^\mathsf{T}) = k$. No such weighted sum exceeds $\sigma_1^2 + \cdots + \sigma_k^2$, and equality is attained by placing all weight on the first k eigendirections. Thus an optimal solution is

$$E = U_k^\mathsf{T}, \qquad D = U_k, \qquad \widehat{x} = U_kU_k^\mathsf{T}x.$$

Define the truncated SVD of X by $X_k = U_k\Sigma_kV_k^\mathsf{T}$. For the training matrix,

$$\widehat{X} = U_kU_k^\mathsf{T}X = U_k\Sigma_kV_k^\mathsf{T} = X_k,$$

and the minimum error is $\sum_{i>k}\sigma_i^2$. This is PCA: the columns of U_k are the principal feature-space directions.

Example 4.5 (A numerical one-dimensional autoencoder)**.** *Take the four centered columns*

$$X = \begin{bmatrix} 2 & -2 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}.$$

Then $XX^\mathsf{T} = \mathrm{diag}(8, 2)$*, so the leading left singular vector is* $u_1 = e_1$ *and the singular values are* $\sqrt{8}, \sqrt{2}$*. With* $k = 1$*,*

$$E = \begin{bmatrix} 1 & 0 \end{bmatrix}, \qquad D = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \qquad Z = EX = \begin{bmatrix} 2 & -2 & 0 & 0 \end{bmatrix}.$$

The reconstruction keeps the horizontal coordinates and discards the vertical pair. Its total squared error is $1^2 + (-1)^2 = 2 = \sigma_2^2$*.*

4.5.2 The Role of the Right Singular Vectors

The matrix U_k acts in the feature space $\mathbb{R}^d$ and therefore acts on any new $x \in \mathbb{R}^d$. Each column of V is a direction in sample-coordinate space $\mathbb{R}^N$; its entries are indexed by training examples, and V depends on that particular training set. It cannot be multiplied by a new feature vector. What the encoder produces on the training set is

$$Z = U_k^{\mathsf{T}} X = \Sigma_k V_k^{\mathsf{T}}.$$

Thus the scaled right singular vectors are the training codes. They depend on the training matrix and are outputs of the encoder, rather than reusable decoder parameters. Their occurrence in the reconstruction follows from

$$U_k Z = U_k \Sigma_k V_k^{\mathsf{T}}.$$

The absence of Σ_k from the encoder and decoder reflects a choice of code scaling: $z = U_k^{\mathsf{T}} x$ carries the singular-value scale in the codes. Assume $\sigma_k > 0$, so that Σ_k is invertible. Removing the singular-value scale gives $z = \Sigma_k^{-1} U_k^{\mathsf{T}} x$, in which case the decoder must be $U_k \Sigma_k$. The reason V is absent is structural; the reason Σ is absent is conventional.

For a centered code matrix Z, define its empirical covariance to be $N^{-1} Z Z^{\mathsf{T}}$. A code is *whitened* when this covariance is the identity. Since $X\mathbf{1} = 0$, the training code matrix under the preceding unnormalized scaling is $Z = V_k^{\mathsf{T}}$ and satisfies $Z\mathbf{1} = 0$; hence it is centered. Its empirical covariance is I_k/N. Unit empirical covariance therefore requires the scale $\sqrt{N}\,\Sigma_k^{-1} U_k^{\mathsf{T}}$. If empirical covariance is instead defined as $(N-1)^{-1} Z Z^{\mathsf{T}}$, the corresponding factor is $\sqrt{N-1}$.

An optimal autoencoder is not unique as a factorization. For any invertible $R \in \mathbb{R}^{k \times k}$,

$$D' = DR^{-1}, \qquad E' = RE \quad \Longrightarrow \quad D'E' = DE.$$

For the orthonormal choice, the product DE is an orthogonal projector.

5 Dual Numbers and Automatic Differentiation

Automatic differentiation is the evaluation of a function together with specified derivatives by applying the chain rule to each elementary operation in its computation. Dual numbers give an algebraic realization of the forward mode; reverse mode will be formulated in terms of computation graphs.

5.1 The Algebra of Dual Numbers

The complex numbers are obtained by introducing a formal element i satisfying $i^2 = -1$. The dual numbers are obtained by introducing a nonzero formal element ε satisfying $\varepsilon^2 = 0$.

Definition 5.1. *The algebra of dual numbers is the two-dimensional real vector space*

$$\mathbb{D} = \{a + b\varepsilon : a, b \in \mathbb{R}\},$$

with multiplication determined by $\varepsilon^2 = 0$. The coefficients a and b are called the real and dual parts, respectively.

Equivalently, $\mathbb{D} = \mathbb{R}[\varepsilon]/(\varepsilon^2)$. Here $\mathbb{R}[\varepsilon]$ is the set of polynomials in ε with the usual addition and multiplication, (ε^2) is the set of polynomials divisible by ε^2, and the quotient identifies two polynomials when their difference is divisible by ε^2. Every resulting class has a unique representative $a + b\varepsilon$.

As a real vector space, $\mathbb{D}$ is the plane with basis $\{1, \varepsilon\}$: the point (a, b) corresponds to $a \cdot 1 + b \cdot \varepsilon$. The real-linear operator $M_\varepsilon : \mathbb{D} \to \mathbb{D}$ defined by $M_\varepsilon(z) = \varepsilon z$ sends (a, b) to $(0, a)$. It has rank one and satisfies $M_\varepsilon^2 = 0$.

Addition and multiplication are

$$(a + b\varepsilon) + (c + d\varepsilon) = (a + c) + (b + d)\varepsilon,$$
$$(a + b\varepsilon)(c + d\varepsilon) = ac + (ad + bc)\varepsilon.$$

The multiplication formula follows from the distributive expansion

$$ac + ad\varepsilon + bc\varepsilon + bd\varepsilon^2 = ac + (ad + bc)\varepsilon,$$

because $bd\varepsilon^2 = 0$. The multiplication is commutative and associative, and $1 + 0\varepsilon$ is the identity. Purely dual products vanish: $(b\varepsilon)(d\varepsilon) = 0$. A nonzero element whose product with some other nonzero element is zero is a *zero divisor*; thus every nonzero $b\varepsilon$ is a zero divisor. An element is *nilpotent* if one of its positive powers is zero; in particular, ε is nilpotent. A *field* is a commutative number system in which every nonzero element is invertible, so $\mathbb{D}$ is not a field. In particular, ε has no multiplicative inverse, since $\varepsilon(c + d\varepsilon) = c\varepsilon$ can never equal 1. In fact, $a + b\varepsilon$ is invertible precisely when $a \neq 0$, and then

$$(a + b\varepsilon)^{-1} = a^{-1} - ba^{-2}\varepsilon.$$

An injective representation by matrices is

$$a + b\varepsilon \longleftrightarrow \begin{bmatrix} a & b \\ 0 & a \end{bmatrix}.$$

Matrix multiplication reproduces dual multiplication. This displayed model acts naturally on row coordinate vectors. With the usual column coordinates in the basis $(1, \varepsilon)$, the multiplication operator is its transpose, $\left[\begin{smallmatrix} a & 0 \\ b & a \end{smallmatrix}\right]$. For the element ε, this is the matrix of the rank-one nilpotent operator M_ε above; it is not a shear. Multiplication by $1 + \varepsilon$ is a shear, represented by $\left[\begin{smallmatrix} 1 & 1 \\ 0 & 1 \end{smallmatrix}\right]$ on row coordinates or its transpose on column coordinates.

5.2 Differentiation by Evaluation

For a differentiable function $f : \mathbb{R} \to \mathbb{R}$, define its dual extension $\widetilde{f} : \mathbb{D} \to \mathbb{D}$ by

$$\widetilde{f}(a + b\varepsilon) = f(a) + bf'(a)\varepsilon.$$

For polynomials and for elementary functions represented locally by convergent power series, this agrees with formal substitution: every term of order two or higher carries ε^2 and vanishes. For a general differentiable function, the displayed equation is the definition of its dual extension; no convergent Taylor series is assumed. We henceforth suppress the tilde when the argument is a dual number.

In particular,

$$f(x + \varepsilon) = f(x) + f'(x)\varepsilon.$$

One evaluation returns both the value and derivative. Division by ε is impossible because ε is not invertible; no difference quotient is formed.

Example 5.2 (A quadratic polynomial)**.** *For* $f(x) = (x + 1)^2$*,*

$$\begin{aligned} f(x + \varepsilon) &= ((x + 1) + \varepsilon)^2 \\ &= (x + 1)^2 + 2(x + 1)\varepsilon + \varepsilon^2 \\ &= f(x) + f'(x)\varepsilon. \end{aligned}$$

The real part is $(x + 1)^2$ *and the dual part is* $2(x + 1)$*.*

Example 5.3 (Powers, reciprocal, and sine)**.** *Direct algebra gives*

$$(x+\varepsilon)^2 = x^2 + 2x\varepsilon, \qquad (x+\varepsilon)^3 = x^3 + 3x^2\varepsilon,$$

and, for $x \neq 0$*,*

$$\frac{1}{x+\varepsilon} = \frac{1}{x} - \frac{1}{x^2}\varepsilon.$$

Using the elementary dual rule for sine,

$$\sin(x+\varepsilon) = \sin x + \cos x\,\varepsilon.$$

The termwise calculation is

$$\begin{aligned}
\sin(x+\varepsilon) &= (x+\varepsilon) - \frac{(x+\varepsilon)^3}{3!} + \frac{(x+\varepsilon)^5}{5!} - \cdots \\
&= x + \varepsilon - \frac{x^3 + 3x^2\varepsilon}{3!} + \frac{x^5 + 5x^4\varepsilon}{5!} - \cdots \\
&= \left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \cdots\right) + \left(1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \cdots\right)\varepsilon \\
&= \sin x + \cos x\,\varepsilon.
\end{aligned}$$

Only the constant and single-ε *terms survive.*

Proposition 5.4 (Sum, product, and quotient rules)**.** *At a fixed point* x*, let* $u = f(x) + f'(x)\varepsilon$ *and* $v = g(x) + g'(x)\varepsilon$*. Dual arithmetic gives*

$$\begin{aligned}
u + v &= (f+g) + (f'+g')\varepsilon, \\
uv &= fg + (f'g + fg')\varepsilon, \\
\frac{u}{v} &= \frac{f}{g} + \frac{f'g - fg'}{g^2}\varepsilon \qquad (g(x) \neq 0),
\end{aligned}$$

where, on the right-hand sides, f, g, f', g' *denote their values at* x*.*

Proof. The first two follow by expansion and $\varepsilon^2 = 0$. For the third, use $v^{-1} = g^{-1} - g'g^{-2}\varepsilon$ and multiply. □

5.3 Composition and the Chain Rule

Applying the dual extensions of g and f successively gives

$$\begin{aligned}
g(x+\varepsilon) &= g(x) + g'(x)\varepsilon, \\
f(g(x+\varepsilon)) &= f(g(x)) + f'(g(x))g'(x)\varepsilon.
\end{aligned}$$

Reading the dual coefficient yields

$$(f \circ g)'(x) = f'(g(x))g'(x).$$

Thus composition automatically multiplies local derivatives.

To solve the equation

$$\cos(t + a\varepsilon) = c + \varepsilon, \qquad c = \cos t,$$

use

$$\cos(t + a\varepsilon) = \cos t - a\sin t\,\varepsilon.$$

Hence $-a\sin t = 1$, so $a = -1/\sin t$, provided $\sin t \neq 0$. If $\sin t = 0$, no such real a exists.

5.4 Several Variables and Forward Mode

Machine-learning models compute functions of many variables, so we now extend dual-number evaluation from a single real variable to maps $F : \mathbb{R}^n \to \mathbb{R}^m$. Let $F = (F_1, \ldots, F_m) : \mathbb{R}^n \to \mathbb{R}^m$ be differentiable. Its *Jacobian matrix* at x is

$$J_F(x) = \left[\frac{\partial F_i}{\partial x_j}(x)\right] \in \mathbb{R}^{m\times n}.$$

For $v \in \mathbb{R}^n$, the *directional derivative* is $D_v F(x) = J_F(x)v$. On first-order dual perturbations, define the componentwise extension $\widetilde{F}$ by

$$\widetilde{F}(x + v\varepsilon) = F(x) + J_F(x)v\,\varepsilon.$$

As in the scalar case, the tilde will be suppressed. To *seed* the direction v in forward mode means to evaluate this extension at $x + v\varepsilon$. The dual coefficient is the Jacobian–vector product $J_F(x)v$. For a scalar function $F : \mathbb{R}^n \to \mathbb{R}$, the gradient is $\nabla F = (\partial F/\partial x_1, \ldots, \partial F/\partial x_n)^\mathsf{T}$. Seeding $v = e_j$ yields $\partial F/\partial x_j$; one seed direction produces one directional derivative, not the entire gradient.

A *computation graph* is a finite directed graph whose nodes are input or intermediate variables and whose directed edges record dependence. It is acyclic: no directed path returns to its starting node. An edge from v to w is labeled by the *local derivative* $\partial w/\partial v$.

5.4.1 A Complete Computation Graph

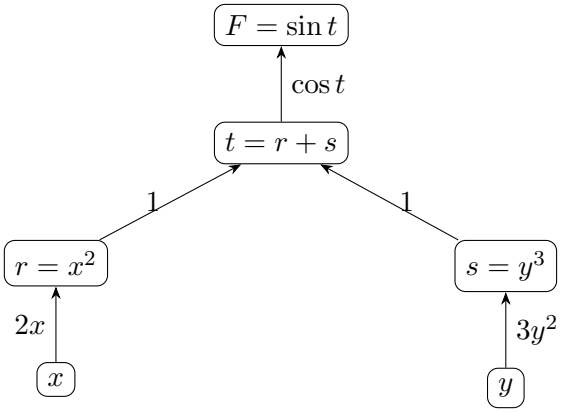
The dual-number rule makes derivative computation automatic. If every elementary operation in a computation is carried out on dual numbers rather than on reals, then, by the chain rule, the dual part of each intermediate is its derivative with respect to the seeded input, accumulated as the computation proceeds. A neural network is exactly such a computation—a composition of affine maps and coordinatewise functions—so evaluating it on a dual input $x + \varepsilon$, with the remaining inputs held real, returns alongside the usual output the partial derivative with respect to the seeded variable, and no separate derivative formula is required. The following example carries this out on a small graph.

Consider the function

$$F(x, y) = \sin(x^2 + y^3), \qquad r = x^2, \quad s = y^3, \quad t = r + s, \quad F = \sin t.$$

Each edge carries a local derivative:

$$\frac{\partial r}{\partial x} = 2x, \quad \frac{\partial s}{\partial y} = 3y^2, \quad \frac{\partial t}{\partial r} = \frac{\partial t}{\partial s} = 1, \quad \frac{\partial F}{\partial t} = \cos t.$$



For the x-partial, seed $x + \varepsilon$ and keep y real:

$$\begin{aligned} r &= (x+\varepsilon)^2 = x^2 + 2x\varepsilon, \\ s &= y^3, \\ t &= x^2 + y^3 + 2x\varepsilon, \\ F &= \sin(x^2+y^3) + 2x\cos(x^2+y^3)\varepsilon. \end{aligned}$$

Therefore

$$F_x = 2x\cos(x^2+y^3).$$

For the y-partial, seed $y + \varepsilon$:

$$\begin{aligned} r &= x^2, \\ s &= (y+\varepsilon)^3 = y^3 + 3y^2\varepsilon, \\ t &= x^2 + y^3 + 3y^2\varepsilon, \\ F &= \sin(x^2+y^3) + 3y^2\cos(x^2+y^3)\varepsilon, \end{aligned}$$

so

$$F_y = 3y^2\cos(x^2+y^3).$$

Finally, seeding $(x + a\varepsilon, y + b\varepsilon)$ returns

$$F(x,y) + \left(2ax + 3by^2\right)\cos(x^2+y^3)\varepsilon,$$

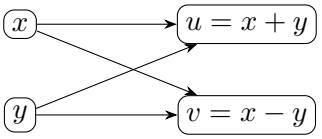
the directional derivative $\nabla F \cdot (a, b)$.

5.4.2 One Infinitesimal per Input

The branching computation graph

$$u = x + y, \qquad v = x - y$$

already shows why one seed may influence several outputs.



Work in the three-dimensional real algebra with basis $\{1, \varepsilon_1, \varepsilon_2\}$ and products $\varepsilon_i\varepsilon_j = 0$ for all $i, j \in \{1, 2\}$. The symbols ε_1 and ε_2 are distinct formal nilpotent generators. Substituting $x + \varepsilon_1$ and $y + \varepsilon_2$ gives

$$u = (x+y) + \varepsilon_1 + \varepsilon_2, \qquad v = (x-y) + \varepsilon_1 - \varepsilon_2.$$

The coefficients record all four entries of this Jacobian.

More generally, use the $(n+1)$-dimensional real algebra with basis $\{1, \varepsilon_1, \ldots, \varepsilon_n\}$ and products $\varepsilon_i\varepsilon_j = 0$ for every i, j. For a differentiable scalar function $f : \mathbb{R}^n \to \mathbb{R}$, a single augmented forward pass then returns

$$f(x) + \sum_{j=1}^{n} \frac{\partial f}{\partial x_j}(x)\varepsilon_j.$$

Every intermediate then stores and updates $1 + n$ real coefficients instead of one. The resulting time and memory therefore grow by a factor proportional to n, just as they do for n separate scalar forward seeds.

5.5 Reverse-Mode Automatic Differentiation

Neural-network training has many input parameters and usually one scalar output, the loss. Forward mode would require one sweep per parameter. Reverse mode instead propagates derivatives from the output toward the inputs.

For each graph variable v, define its adjoint

$$\bar{v} = \frac{\partial L}{\partial v}.$$

Perform a forward sweep and store the intermediate values. Set $\bar{L} = 1$ and initialize every other adjoint to zero. A topological order lists every node after all nodes on which it depends; traverse the nodes in the reverse of such an order. Whenever v directly feeds q, add the indicated contribution to the current value of $\bar{v}$:

$$\bar{v} \mathrel{+}= \bar{q}\frac{\partial q}{\partial v}.$$

Here $a \mathrel{+}= b$ means that a is replaced by $a + b$. Equivalently,

$$\bar{v} = \sum_{q:v\to q} \bar{q}\frac{\partial q}{\partial v}.$$

The sum is essential at fan-out: every downstream path contributes. This recurrence is written for scalar graph nodes; vector nodes use the same principle with Jacobian transposes.

For a map with Jacobian $J \in \mathbb{R}^{m\times n}$, forward mode computes the Jacobian–vector product Jv. Given an output vector $u \in \mathbb{R}^m$, reverse mode computes the column vector $J^{\mathsf{T}}u$, whose transpose is the vector–Jacobian product $u^{\mathsf{T}}J$. For a scalar output, $u = 1$, so one reverse sweep returns the entire gradient. If evaluation uses C scalar arithmetic operations and each local derivative has bounded cost, the reverse sweep uses at most a constant multiple of C operations. It must additionally retain or recompute the forward intermediate values.

5.5.1 The Running Graph in Reverse

For $F = \sin(x^2 + y^3)$, a reverse sweep gives

$$\begin{aligned}
&\bar{F} = 1,\\
&\bar{t} = \bar{F}\frac{\partial F}{\partial t} = \cos t,\\
&\bar{r} = \bar{t}\frac{\partial t}{\partial r} = \cos t, \qquad && \bar{s} = \bar{t}\frac{\partial t}{\partial s} = \cos t,\\
&\bar{x} = \bar{r}\frac{\partial r}{\partial x} = 2x\cos t, && \bar{y} = \bar{s}\frac{\partial s}{\partial y} = 3y^2\cos t.
\end{aligned}$$

With $t = x^2 + y^3$, these are exactly the two gradient components. This reproduces both partial derivatives in one backward sweep.

5.5.2 Fan-out Example

Let

$$y = x_1x_2 + \sin x_1, \qquad v_1 = x_1x_2, \quad v_2 = \sin x_1, \quad y = v_1 + v_2.$$

At $(x_1, x_2) = (a, b)$ the forward values are $v_1 = ab$, $v_2 = \sin a$, and $y = ab + \sin a$. Backward propagation gives

$$\bar{y} = 1, \quad \bar{v}_1 = \bar{v}_2 = 1, \quad \bar{x}_2 = \bar{v}_1 x_1 = a,$$

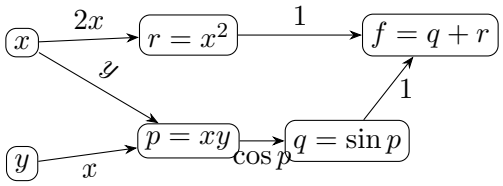
and, because x_1 has two outgoing edges,

$$\bar{x}_1 = \underbrace{\bar{v}_1 x_2}_{\text{via } v_1} + \underbrace{\bar{v}_2 \cos x_1}_{\text{via } v_2} = b + \cos a.$$

5.5.3 A Numerical Backpropagation Example

Let

$$f(x, y) = \sin(xy) + x^2, \quad p = xy, \quad q = \sin p, \quad r = x^2, \quad f = q + r.$$



At $(x, y) = (2, 1)$ the forward sweep is

variable	value	local derivatives
x	2	$--$
y	1	$--$
$p = xy$	2	$p_x = y = 1, \quad p_y = x = 2$
$q = \sin p$	$\sin 2 \approx 0.9093$	$q_p = \cos 2 \approx -0.4161$
$r = x^2$	4	$r_x = 2x = 4$
$f = q + r$	$4 + \sin 2 \approx 4.9093$	$f_q = f_r = 1$

Equivalently, the weighted adjacency table of direct local derivatives has precisely these nonzero entries:

$$x \to p : 1, \quad x \to r : 4, \quad y \to p : 2, \quad p \to q : \cos 2, \quad q \to f : 1, \quad r \to f : 1.$$

All other distinct direct-edge entries are zero. This is a table of local edge derivatives, not of total derivatives between every pair of variables. The backward sweep is

$$\begin{aligned}
&\bar{f} = 1, \\
&\bar{q} = 1, && \bar{r} = 1, \\
&\bar{p} = \cos 2, \\
&\bar{y} = \bar{p}\, x = 2 \cos 2, \\
&\bar{x} = \underbrace{\bar{p}\, y}_{\text{through } p} + \underbrace{\bar{r}\,(2x)}_{\text{through } r} = \cos 2 + 4.
\end{aligned}$$

Hence

$$\nabla f(2, 1) = (\cos 2 + 4, 2 \cos 2) \approx (3.5839, -0.8323).$$

The corresponding rounded values are $(3.58, -0.83)$. Direct differentiation verifies the calculation:

$$f_x = y \cos(xy) + 2x, \qquad f_y = x \cos(xy),$$

which at $(2, 1)$ gives the same two components.

5.6 Backpropagation Through a Neural-Network Layer

The graph rules extend to a matrix-valued layer. For

$$z = Ax + b,$$

let $\bar{z} = \partial L/\partial z$ be the column vector of partial derivatives of L with respect to the coordinates of z. The symbol d denotes the first-order differential. Thus

$$dz = (dA)x + A\,dx + db, \qquad dL = \bar{z}^{\mathsf{T}} dz.$$

For a matrix variable A, define its gradient by the Frobenius pairing

$$dL = \operatorname{tr}\left(\left(\frac{\partial L}{\partial A}\right)^{\mathsf{T}} dA\right) + \bar{x}^{\mathsf{T}} dx + \left(\frac{\partial L}{\partial b}\right)^{\mathsf{T}} db.$$

Comparing the coefficients of dA, dx, and db yields

$$\boxed{\bar{x} = A^{\mathsf{T}}\bar{z}}, \qquad \boxed{\frac{\partial L}{\partial b} = \bar{z}}, \qquad \boxed{\frac{\partial L}{\partial A} = \bar{z}x^{\mathsf{T}}}.$$

For column vectors u and v, the matrix uv^{T} is their *outer product*. The weight gradient above is therefore a rank-one outer product for a single example, and the gradients from several examples add these contributions. For several examples at once, let $X \in \mathbb{R}^{d\times N}$, $A \in \mathbb{R}^{k\times d}$, $b \in \mathbb{R}^k$, and $\mathbf{1} \in \mathbb{R}^N$. Define $Z = AX + b\mathbf{1}^{\mathsf{T}} \in \mathbb{R}^{k\times N}$, $\bar{Z} = \partial L/\partial Z \in \mathbb{R}^{k\times N}$, and $\bar{X} = \partial L/\partial X \in \mathbb{R}^{d\times N}$. Then

$$Z = AX + b\mathbf{1}^{\mathsf{T}}, \qquad \bar{X} = A^{\mathsf{T}}\bar{Z}, \qquad \frac{\partial L}{\partial A} = \bar{Z}X^{\mathsf{T}}, \qquad \frac{\partial L}{\partial b} = \bar{Z}\mathbf{1}.$$

If $h = \sigma(z)$ coordinatewise, then

$$\bar{z} = \bar{h} \odot \sigma'(z).$$

Here $\odot$ denotes entrywise, or Hadamard, multiplication. Alternating these two local rules backward through a feedforward network is backpropagation. It produces the $\nabla L(\theta)$ required by the update that opened the notes:

$$\theta \leftarrow \theta - \eta \nabla L(\theta).$$

5.7 Comparison with Other Differentiation Methods

In *symbolic differentiation*, algebraic rules are applied to construct a new expression for a derivative. A *finite-difference* approximation uses, for example, $[f(x + h) - f(x)]/h$ for a nonzero real step h. Automatic differentiation instead evaluates exact chain-rule identities along a computation graph.

- It is exact up to floating-point roundoff, the error introduced by finite precision: there is no finite step size and hence no truncation error from discarding higher-order powers of that step.
- Each elementary operation supplies a local derivative rule, and composition applies the chain rule.
- Forward mode successively applies Jacobians to an input direction, whereas reverse mode successively applies transposed Jacobians to an output adjoint. Forward mode requires one sweep per input direction, whereas reverse mode requires one sweep per output direction; the latter is therefore suited to a scalar loss.

- Reverse mode normally stores the forward intermediate values in order to avoid recomputing them during the backward sweep.

At a nondifferentiable elementary operation, such as ReLU at zero, the classical derivative does not exist. For a convex function g, a *subgradient* at x is a number s such that $g(y) \geq g(x) + s(y - x)$ for every y. Software must adopt a stated convention at such a point, commonly choosing one subgradient; a conditional expression is differentiated along the branch that was evaluated. These are modeling and implementation conventions, not consequences of dual-number algebra.

5.8 Practice Problems

1. Reproduce the seven sign patterns in the arrangement of three lines using your own interior points. Prove directly that the missing pattern $(-, +, +)$ is impossible.
2. For a $2 \times N$ layer, identify w_{ij} in the pipe picture and prove from the column picture that its range is a point, line, or plane.
3. For $f(x) = \operatorname{sign}(\theta^{\mathsf{T}} x)$, explain why positive but not negative rescaling leaves the function fixed. Describe the quotient.
4. Derive $BA = \sum_j b_j a_j^{\mathsf{T}}$ and use it to give two proofs that $\operatorname{rank}(BA) \leq k$.
5. Compute a full SVD of $\left[\begin{smallmatrix} 3 & 0 \\ 0 & 2 \\ 0 & 0 \end{smallmatrix}\right]$ and describe the image of the unit circle.
6. Prove the Pythagorean identity for $DD^{\mathsf{T}}X$ when $D^{\mathsf{T}}D = I$. Then derive the linear-autoencoder objective in trace form.
7. For $X = U\Sigma V^{\mathsf{T}}$, verify both $U_k^{\mathsf{T}} X = \Sigma_k V_k^{\mathsf{T}}$ and $U_k U_k^{\mathsf{T}} X = X_k$. Explain the dimensions of every factor.
8. Compute $(a + b\varepsilon)^{-1}$ directly and use it to recover the quotient rule.
9. Evaluate $(x^2 + 1)^3$ at $x + \varepsilon$ using only dual arithmetic.
10. Perform both forward seeds and the reverse sweep for $F(x, y) = e^{xy} + x/y$ at a point with $y \neq 0$.
11. Starting from differentials, derive the reverse rules for $z = Ax + b$ and $h = \operatorname{ReLU}(z)$.

References

[1] G. Strang, *Introduction to Linear Algebra*, 5th ed., Wellesley–Cambridge Press, 2016.

[2] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 4th ed., Johns Hopkins University Press, 2013.

[3] W. K. Clifford, "Preliminary Sketch of Biquaternions," *Proceedings of the London Mathematical Society* s1-4 (1873), 381–395.

[4] I. M. Yaglom, *Complex Numbers in Geometry*, Academic Press, 1968.

[5] R. E. Wengert, "A Simple Automatic Derivative Evaluation Program," *Communications of the ACM* 7(8) (1964), 463–464.

[6] S. Linnainmaa, "Taylor Expansion of the Accumulated Rounding Error," *BIT Numerical Mathematics* 16(2) (1976), 146–160.

[7] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. thesis, Harvard University, 1974.

[8] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, Lecture Notes in Computer Science 120, Springer, 1981.

[9] A. Griewank and A. Walther, *Evaluating Derivatives*, 2nd ed., SIAM, 2008.

[10] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature* 323 (1986), 533–536.

[11] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic Differentiation in Machine Learning: A Survey," *Journal of Machine Learning Research* 18(153) (2018), 1–43.

[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.