Vectors in Machine Learning: Complete Guide

Introduction to Vectors

A **vector** is a mathematical object that has both **magnitude** (size/length) and **direction**. Unlike a scalar (a single number), a vector encodes multiple pieces of information simultaneously.

Intuitive Understanding

Think of a vector as an arrow:

- The **length of the arrow** is the magnitude
- The **direction the arrow points** is the direction
- The **starting point** is the origin

In everyday life, vectors describe things that have direction and magnitude:

- Wind velocity: "15 m/s northeast"
- Force: "10 Newtons upward"
- Displacement: "5 miles east"

Mathematical Representation

A vector in n-dimensional space is written as an ordered list of numbers (components):

$$v = [v_1, v_2, v_3, \ldots, v_n]$$

Or in column form:

$$v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix}$$

Example: A 2D vector v = [3, 4] means:

- Move 3 units in the x-direction
- Move 4 units in the y-direction

Example: A 3D vector v = [1, -2, 3] means:

- Move 1 unit in x-direction
- Move -2 units (backward) in y-direction
- Move 3 units in z-direction

Representation of Vectors in Machine Learning

Why Vectors Are Central to ML

Machine learning models operate on vectors. Every piece of data is represented as a vector.

Feature Vectors

A **feature vector** represents a single data point with all its characteristics as components.

Example: Predicting house prices

```
house = [area_sqft, num_bedrooms, age_years, price]
      = [2500, 4, 10, 450000]
```

Each house is a 4-dimensional vector (after adding price for the target). During training, we have thousands of such vectors, one per house.

Example: Image classification Each image (say 28×28 pixels) can be flattened into a single vector of 784 dimensions (28 × 28 = 784 pixel values).

```
image_vector = [pixel₁, pixel₂, pixel₃, ..., pixel₇₈₄]
             = [0.2, 0.8, 0.1, ..., 0.9]
```

Example: Text processing (word embeddings) Words are represented as vectors (embeddings) where each dimension captures semantic meaning.

```
king = [0.2, 0.5, -0.3, 0.1, ...]      (300 dimensions)
queen = [0.3, 0.4, -0.2, 0.1, ...]     (similar but different)
man = [0.1, 0.6, -0.4, 0.2, ...]
woman = [0.2, 0.5, -0.3, 0.2, ...]
```

Interestingly: king - man ≈ queen - woman (vectors capture relationships!)

Parameter Vectors in Models

Neural networks store parameters as vectors:

$$\text{weights} = [w_1, w_2, w_3, ..., w_n]$$
$$\text{biases} = [b_1, b_2, b_3, ..., b_m]$$

During training, we adjust these vectors to minimize loss.

Gradient Vectors

The gradient (used in optimization) is itself a vector:

$$\nabla f = [\partial f/\partial x_1, \partial f/\partial x_2, \partial f/\partial x_3, ..., \partial f/\partial x_n]$$

Each component shows how much changing that parameter affects the loss.

Scaling Vectors (Scalar Multiplication)

Scaling (or scalar multiplication) means multiplying a vector by a single number (scalar).

Mathematical Definition

If $v = [v_1, v_2, ..., v_n]$ and c is a scalar, then:

$$c \cdot v = [c \cdot v_1, c \cdot v_2, ..., c \cdot v_n]$$

Geometric Interpretation

Scaling changes the magnitude of the vector but not its direction (unless the scalar is negative, which also reverses direction).

Example:

$$v = [2, 3]$$
$$2 \cdot v = [4, 6] \quad \text{(twice as long, same direction)}$$
$$-1 \cdot v = [-2, -3] \quad \text{(same length, opposite direction)}$$
$$0.5 \cdot v = [1, 1.5] \quad \text{(half as long, same direction)}$$

If you plot these, they all point in the same or opposite directions, just different lengths.

Applications in ML

1. Learning rates in gradient descent:

$$w_{new} = w_{old} - \alpha \cdot \nabla f(w_{old})$$

The learning rate α is a scalar that scales the gradient. Larger α means bigger steps toward lower loss.

2. Regularization: Scaling parameter vectors by a factor less than 1:

```
w_regularized = 0.99·w_original
```

This encourages the model to use smaller weights, preventing overfitting.

3. Normalization: Scaling vectors to have unit length (discussed later with norms).

Vector Addition and Geometric Interpretation

Vector addition combines two vectors by adding their corresponding components.

Mathematical Definition

If $u = [u_1, u_2, ..., u_n]$ and $v = [v_1, v_2, ..., v_n]$, then:

$$u + v = [u_1 + v_1, u_2 + v_2, ..., u_n + v_n]$$

Geometric Interpretation: Head-to-Tail

When visualizing vector addition in 2D or 3D:

1. Draw the first vector u from the origin
2. Draw the second vector v starting from the head (tip) of u
3. The result u + v is drawn from the origin to the head of v

This forms a triangle. The resulting vector is the diagonal of the parallelogram formed by u and v.

Example:

```
u = [2, 1]
v = [1, 3]
u + v = [3, 4]
```

Geometrically: if you go 2 units right and 1 unit up, then 1 more right and 3 more up, you end at [3, 4].

Applications in ML

1. Combining multiple sources of information: In ensemble models, predictions from different models are added:

```
final_prediction = model1_prediction + model2_prediction + model3_prediction
```

2. Feature engineering: New features can be created by adding existing features:

```
combined_feature = age_feature + experience_feature
```

3. Bias terms in neural networks: When computing a neuron's output:

```
output = weights·inputs + bias
```

This is addition of a weighted input vector and a bias vector.

Vector Subtraction

Vector subtraction is defined as adding the negative:

$$u - v = u + (-1)\cdot v = [u_1 - v_1, u_2 - v_2, ..., u_n - v_n]$$

Geometric Interpretation

Subtraction points from the tip of v to the tip of u.

Example:

```
u = [5, 3]
v = [2, 1]
u - v = [3, 2]
```

If u and v are positions, u - v is the displacement vector from v to u.

Applications in ML

1. Error vectors:

```
error = predicted_values - actual_values
```

The error vector shows where predictions went wrong.

2. Differences between data points:

```
difference = point_a - point_b
```

The distance between points is the magnitude of this difference vector.

3. Gradient direction: In gradient descent, we move opposite the gradient:

```
w_new = w_old - α·∇f(w_old)
```

This is vector subtraction: subtract a scaled gradient from current parameters.

Dot Product and Its Meaning in ML

The **dot product** (also called inner product or scalar product) takes two vectors and produces a single number.

Mathematical Definition

For vectors $u = [u_1, u_2, ..., u_n]$ and $v = [v_1, v_2, ..., v_n]$:

$$u \cdot v = u_1v_1 + u_2v_2 + ... + u_nv_n$$

Multiply corresponding components and sum all products.

Example:

$u = [2, 3, 1]$
$v = [4, 1, 5]$
$u \cdot v = (2)(4) + (3)(1) + (1)(5) = 8 + 3 + 5 = 16$

Geometric Interpretation

The dot product encodes the relationship between two vectors' directions:

$$u \cdot v = ||u|| \cdot ||v|| \cdot \cos(\theta)$$

Where θ is the angle between the vectors, and $||u||$ denotes the magnitude (length).

What this means:

- If u and v point in the **same direction** ($\theta = 0°$): $\cos(0°) = 1$, so $u \cdot v$ is **large and positive**
- If u and v are **perpendicular** ($\theta = 90°$): $\cos(90°) = 0$, so $u \cdot v = \mathbf{0}$
- If u and v point in **opposite directions** ($\theta = 180°$): $\cos(180°) = -1$, so $u \cdot v$ is **large and negative**

Dot Product Measures Similarity

The dot product is a **similarity metric**: it quantifies how much two vectors "agree" in direction.

Examples:

- $u \cdot v = 100$: vectors are similar (small angle between them)
- $u \cdot v = 0$: vectors are orthogonal (perpendicular, independent)
- $u \cdot v = -100$: vectors point in opposite directions

Applications in ML

1. Neural network forward pass: Each neuron computes:

```
z = w · x + b
```

Where w is the weight vector, x is the input vector, and b is the bias. The dot product $w \cdot x$ combines all weighted inputs.

2. Similarity in recommendation systems: If users are represented as vectors of preferences, we find similar users using:

```
similarity(user_a, user_b) = user_a · user_b
```

Larger dot product means more similar preferences.

3. Attention mechanisms in transformers: Attention is computed using dot products of query and key vectors:

```
attention_weights = (query · key) / √(dimension)
```

This determines which parts of the input to focus on.

4. Cosine similarity: Normalizing the dot product:

```
cosine_similarity = (u · v) / (||u|| · ||v||)
```

This measures similarity on a scale from -1 to 1, independent of vector magnitude.

Vector Projections

A **projection** finds the component of one vector in the direction of another.

Geometric Interpretation

Imagine shining a light perpendicular to vector v. The shadow cast by vector u on v is the **projection of u onto v**.

Mathematical Definition

The projection of u onto v is:

$$\text{proj}_v(u) = (u \cdot v / \|v\|^2) \cdot v$$

This gives a new vector that lies along v and represents how much u "points in the direction of v."

The **scalar projection** (magnitude of the projection) is:

$$\text{comp}_v(u) = (u \cdot v) / \|v\| = \|u\| \cdot \cos(\theta)$$

Geometric Example

$u = [3, 4]$ (vector in 2D)
$v = [1, 0]$ (unit vector along x-axis)

$u \cdot v = 3(1) + 4(0) = 3$
$\|v\|^2 = 1^2 + 0^2 = 1$

$\text{proj}_v(u) = (3/1) \cdot [1, 0] = [3, 0]$

The projection of [3, 4] onto the x-axis is [3, 0]—the x-component of the vector.

Applications in ML

1. Dimensionality reduction: Principal Component Analysis (PCA) projects high-dimensional data onto lower-dimensional subspaces (defined by principal components).

2. Feature extraction: Projecting raw data onto learned feature directions to create meaningful representations.

3. Decomposing vectors: Breaking down a vector into components along different directions:

$$u = \text{proj}_v(u) + (u - \text{proj}_v(u))$$

The first term is the component along v, the second is the component perpendicular to v.

Orthogonality Explained Visually

Two vectors are **orthogonal** (perpendicular) if their dot product is zero.

Mathematical Definition

u and v are orthogonal if and only if:

$$u \cdot v = 0$$

From the geometric formula $u \cdot v = ||u|| \cdot ||v|| \cdot \cos(\theta)$, this happens when $\cos(\theta) = 0$, which occurs at $\theta = 90°$.

Geometric Interpretation

Orthogonal vectors point in completely independent directions. They share no component with each other.

Visual example in 2D:

$u = [1, 0]$ (points right)
$v = [0, 1]$ (points up)

$u \cdot v = 1(0) + 0(1) = 0$

They form a 90° angle.

Orthonormal Vectors

A set of vectors is **orthonormal** if:

1. All vectors are orthogonal to each other
2. All vectors have length 1 (unit length)

The standard basis vectors are orthonormal:

$e_1 = [1, 0, 0]$
$e_2 = [0, 1, 0]$
$e_3 = [0, 0, 1]$

Applications in ML

1. Feature independence: Orthogonal features carry independent information. This is desirable because each feature contributes unique information to predictions.

2. Gram-Schmidt orthogonalization: A technique to convert correlated features into orthogonal features while preserving information.

3. Rotation matrices: In computer vision, rotation matrices have orthogonal rows and columns. Applying them preserves distances and angles (important for data augmentation).

4. Regularization intuition: L2 regularization in neural networks encourages weights to be orthogonal to the data distribution's principal directions, improving generalization.

Cross Product and Its Applications

The **cross product** (only defined for 3D vectors) produces a new vector perpendicular to both input vectors.

Mathematical Definition

For $u = [u_1, u_2, u_3]$ and $v = [v_1, v_2, v_3]$:

$$u \times v = [u_2v_3 - u_3v_2, u_3v_1 - u_1v_3, u_1v_2 - u_2v_1]$$

Geometric Interpretation

The cross product produces:

1. A **direction**: perpendicular to both u and v (determined by the right-hand rule)
2. A **magnitude**: $||u \times v|| = ||u|| \cdot ||v|| \cdot \sin(\theta)$, proportional to the area of the parallelogram formed by u and v

Visualization: If u and v lie on a table, u × v points straight up (or down) from the table—perpendicular to the plane they define.

Key Properties

- **Orthogonality**: u × v is perpendicular to both u and v
- **Anticommutativity**: u × v = -(v × u) (direction reverses if order swaps)
- **Parallel vectors**: If u and v are parallel, u × v = 0 (zero area parallelogram)

Applications in ML

1. 3D computer vision: Computing surface normals (perpendicular to surfaces) for 3D object recognition and reconstruction.

2. Physics-informed neural networks: Encoding rotational dynamics in neural networks for robotics or physics simulations.

3. Geometric deep learning: Handling 3D point clouds and mesh data where cross products help compute local geometry.

4. Feature generation: Creating interaction features in 3D feature spaces (though less common than 2D interactions).

Vector Norms: L1, L2, and Beyond

A **norm** measures the length or magnitude of a vector. Different norms are useful for different purposes.

L2 Norm (Euclidean Norm)

The most common norm, representing the standard "length" of a vector:

$$||v||_2 = \sqrt{(v_1^2 + v_2^2 + ... + v_n^2)}$$

This is the distance from the origin to the point represented by the vector.

Example:

$v = [3, 4]$
$||v||_2 = \sqrt{(3^2 + 4^2)} = \sqrt{(9 + 16)} = \sqrt{25} = 5$

L1 Norm (Manhattan Norm)

The sum of absolute values:

$$||v||_1 = |v_1| + |v_2| + ... + |v_n|$$

Also called "Manhattan distance" because it's like counting blocks traveled in a grid.

Example:

$v = [3, 4]$
$||v||_1 = |3| + |4| = 7$

Geometrically: if you travel along a grid (only horizontal/vertical), you cover distance 7. But straight-line distance (L2) is only 5.

L∞ Norm (Maximum Norm)

The largest absolute component:

$$||v||_\infty = \max(|v_1|, |v_2|, ..., |v_n|)$$

Example:

```
v = [3, 4]
||v||∞ = max(|3|, |4|) = 4
```

Lp Norm (General Form)

All norms fit the pattern:

$$||v||_p = (|v_1|^p + |v_2|^p + ... + |v_n|^p)^{(1/p)}$$

Different values of p give different norms.

Visual Comparison

In 2D, all points at distance 1 from origin form different shapes:

- **L2 norm**: a circle (Euclidean)
- **L1 norm**: a diamond (Manhattan)
- **L∞ norm**: a square (Chebyshev)

Applications in ML

1. Distance metrics for clustering:

- K-means typically uses L2 (Euclidean) distance
- K-medoids can use L1 (Manhattan) distance, more robust to outliers

2. Regularization:

L2 Regularization (Ridge):

$$\text{Loss} = \text{prediction\_error} + \lambda \cdot ||w||_2^2$$

Penalizes large weights, encourages smaller values.

L1 Regularization (Lasso):

$$\text{Loss} = \text{prediction\_error} + \lambda \cdot ||w||_1$$

More aggressive: drives small weights to exactly zero, performing automatic feature selection.

3. Normalization: Converting vectors to unit length:

$$v_{normalized} = v / ||v||_2$$

Used in cosine similarity, attention mechanisms, and many deep learning applications.

4. Constraint satisfaction: Some algorithms require vectors with bounded norms (batch normalization, weight normalization).

5. Robustness: L1 norm is more robust to outliers than L2. When data contains extreme values, L1 distance is often more stable.

Distance Measures Between Vectors

Vectors enable quantifying distance between data points—fundamental to ML.

Euclidean Distance (L2)

The straight-line distance between two points u and v:

$$d(u, v) = ||u - v||_2 = \sqrt{((u_1-v_1)^2 + (u_2-v_2)^2 + ... + (u_n-v_n)^2)}$$

Example:

$u = [1, 2]$
$v = [4, 6]$
$u - v = [-3, -4]$
$d(u, v) = \sqrt{((-3)^2 + (-4)^2)} = \sqrt{(9 + 16)} = 5$

Manhattan Distance (L1)

Grid-based distance:

$$d(u, v) = ||u - v||_1 = |u_1-v_1| + |u_2-v_2| + ... + |u_n-v_n|$$

Example:

$u = [1, 2]$
$v = [4, 6]$
$d(u, v) = |1-4| + |2-6| = 3 + 4 = 7$

Cosine Similarity

Measures angle between vectors, independent of magnitude:

$$\text{similarity} = (u \cdot v) / (\|u\|_2 \cdot \|v\|_2)$$

Ranges from -1 (opposite) to 1 (identical direction).

Used when: Direction matters more than magnitude (e.g., comparing word embeddings).

Hamming Distance

For binary vectors, counts positions where they differ:

$$d(u, v) = \text{number of positions where } u_i \neq v_i$$

Used when: Comparing categorical features or binary representations.

Applications in ML

1. K-Nearest Neighbors: Classifies a point based on the K nearest neighbors:

$$\text{distance\_to\_training\_point} = \|\text{test\_point} - \text{training\_point}\|$$

2. Clustering: K-means assigns points to nearest cluster center:

$$\text{argmin}_c \|\text{point} - \text{cluster\_center}_c\|^2$$

3. Retrieval and recommendation: Finding similar items by computing distances in representation space.

4. Anomaly detection: Points far from normal data (high distance to typical points) are flagged as anomalies.

5. Similarity search: Finding nearest neighbors in embedding space for semantic search, recommendation systems, etc.

Summary: Vectors as the Language of ML

Vectors are the fundamental language of machine learning:

- **Representation**: Data points, features, parameters, and gradients are all vectors
- **Operations**: Addition, subtraction, scaling combine vectors; dot product measures similarity; norms measure magnitude
- **Geometry**: Vectors provide intuitive geometric understanding of high-dimensional spaces
- **Distance**: Computing distances between vectors enables clustering, classification, and similarity search

- **Optimization**: Gradients are vectors pointing toward improvement; we follow them downhill

Understanding vectors deeply transforms ML from symbol manipulation to geometric intuition: you're navigating high-dimensional spaces, measuring distances, finding similar points, and minimizing loss functions in directions revealed by derivatives.

This geometric perspective is essential for understanding why algorithms work, debugging problems, and inventing new techniques.