Geometry and linear algebra for IN3050/IN4050

Jan Tore Lønning

Second edition—February 2021

Abstract

Machine learning (ML) builds on knowledge from several fields, in particular probabilities and linear algebra. We will in IN3050/IN4050 keep probabilities at a minimum. But, as we will be interested in the algorithmic properties of ML, we need some background in linear algebra, vectors, and matrices. The IN3050/IN4050 students come from several study programs and have different mathematical backgrounds. Many books on ML contains some first chapters, or appendices, with background knowledge in topics like probabilities and linear algebra. Marsland, *Machine Learning* contains e.g., an appendix on Python and NumPy, but assumes familiarity with linear algebra from the reader, cf. "You've probably seen this in high school or somewhere . . . " (p.50). The following pages are meant as an appendix on linear algebra for Marsland's book. Some of you probably know everything already, for others it may serve as a fresh up, and for some as an introduction. Of course, this cannot replace a more proper education on linear algebra, but it will hopefully be of some help.

This was first written in 2020. It contained several typos and unclear statments. We have tried to correct them, but there are sure remaining faults in this edition. Please report them or ask if you have questions.

1 Cartesian coordinates

1.1 Points in the plane

We begin with the basics, points in the plane. By introducing two perpendicular axes, we can refer to a point in the plane with a pair of real numbers, cf. figure 1.

We will write (x, y) for a general point in the plane. In these days, we are so familiar with this connection between algebra and geometry that we take it for granted. This connection is, however, far from self-evident, and it was a major achievement by René Descartes (1596-1650) to establish it. By the way, Descartes also used this discovery to study optique and how light rays were processed by the eye, and speculated how this could further be processed by the brain. He was thereby one of the forerunners of AI. But that is another story.

Remember Pythagoras' theorem; in a right triangle (one where one of the angles is a right angle), we have the following relationship between the lengths of the hypothenuse (the side opposite the right angle), c, and the two other sides, a and b (called the *catheti*, plural of *cathetus*, figure 2).

$$a^2 + b^2 = c^2$$

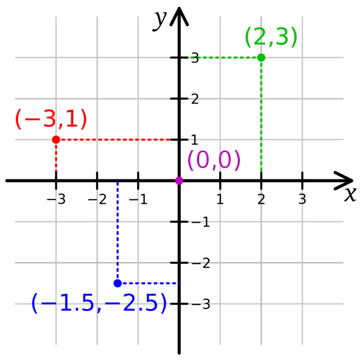

Figure 1: Points in the plane (Source: Wikipedia)

From this, we can define a distance between two points in the Cartesian plane by the following formula, cf., figure 3.

$$d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{1}$$

It does not matter in which order we insert the two points, as we remember $(-5)^2 = 5^2 = 25$, and $\sqrt{(-5)^2} = \sqrt{25} = |5| = 5$, where $|a|$, the absolute value of a, is defined by $|a| = a$ for $a \geq 0$ and $|a| = -a$ for $a < 0$.

The distance function, as we have defined it here, is the common (or "normal") way to define distance in the Cartesian plane. In the machine learning literature, it is often called the L^2- or *l2-distance*. It is possible to define other distance metrics in the plane. Another metric also used in machine learning is the *Manhattan* or L^1-distance defined by the following formula.

$$d_1((x_1, y_1), (x_2, y_2)) = |x_2 - x_1| + |y_2 - y_1| \tag{2}$$

Why do you think it is called *Manhattan distance*?

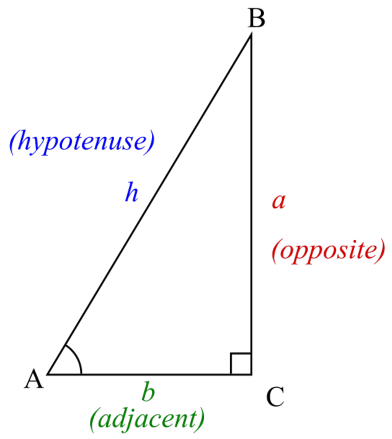

Figure 2: Right triangle (Wikipedia)

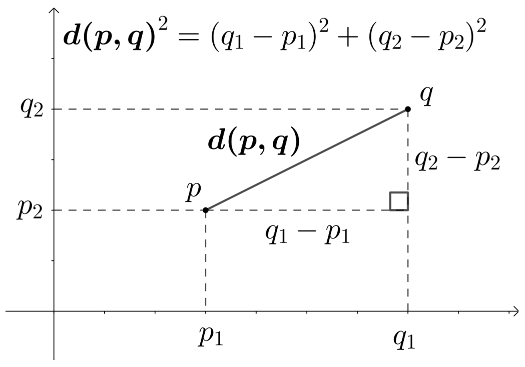

Figure 3: Distance in the plane (Wikipedia)

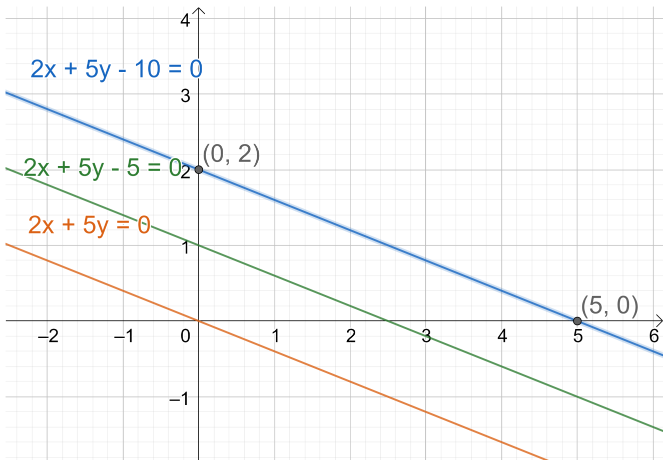

Figure 4: Parallel lines (Made with GeoGebra)

1.2 Lines in the plane

Given three real numbers, a, b, and c, where at least one of a or b does not equal 0. all points (x, y) satisfying

$$ax + by + c = 0 \tag{3}$$

will lay on a straight line. If $b \neq 0$, we can consider $x_1 = 0$ and find the intercept with the y-axis, $y_1 = \frac{-c}{b}$. Similarly, if $a \neq 0$, we find the intercept with the x-axis, $x_2 = \frac{-c}{a}$. For example, take the line $2x + 5y - 10 = 0$. We find the intercepts, $(0, 2)$ and $(5, 0)$; see figure 4. If $b \neq 0$, we can find the slope of the line, as $m = -\frac{a}{b}$. For the example, the slope $m = -\frac{2}{5}$ is negative, which means that the line is leaning downwards to the right.

So far, we have taken the equation for the line as a starting point. Alternatively, we could start with two points. For any two different points in the plane, there is one and exactly one line passing through the two points. For the two points (x_0, y_0) and (x_1, y_1), where $x_0 \neq x_1$, the line consists of all points (x, y) satisfying the following formula.

$$\frac{y - y_0}{x - x_0} = \frac{y_1 - y_0}{x_1 - x_0} \tag{4}$$

It is straightforward to transform this to the form $ax + by + c = 0$ and express a, b and c in terms of x_0, x_1, y_0, y_1. In particular, we can calculate the slope of the line

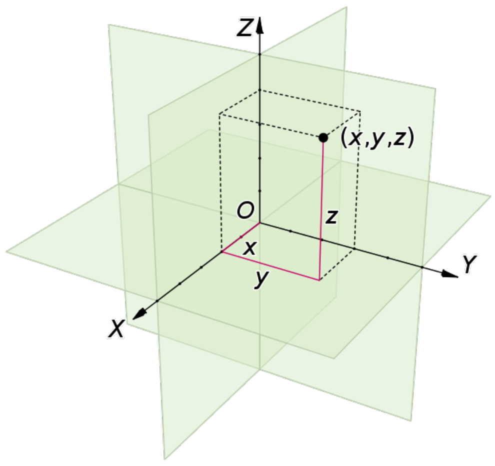

Figure 5: Points in 3D space (Wikipedia)

to be $m = \frac{y_1 - y_0}{x_1 - x_0}$. This formula also works for lines passing through the origin. (By the way, *origin* is the point $(0, 0)$ where the two axes cross each other, remember?)

Two lines in the plane which do not cross each other are said to be *parallel*. If we keep a and b the same, but change c, we get parallel lines. For $c = 0$, we get a line through the origin. What happens if $a = 0$ or $b = 0$; for example, which line is $5y - 10 = 0$ or $2x - 10 = 0$?

1.3 3D and higher dimensions

Just like we can locate points in the plane with respect to two axes, points in space can be located related to three axes. The three axes must meet in a point, the origin, and be pair-wise perpendicular. As in the two-dimensional case, each axis has to be oriented and the three axes use the same unit of length. Any point can then be described with three numbers, its coordinates with respect to the three axes, (x, y, z).

The definition of distance generalizes to the 3D space. For example, in figure 5, let c be the line segment from $(0, 0, 0)$ to $(x, y, 0)$ and d the line segment from $(0, 0, 0)$ to (x, y, z). By Pythagoras' theorem, we see that $d^2 = c^2 + z^2$ and $c^2 = x^2 + y^2$, hence $d^2 = x^2 + y^2 + z^2$. We can define the general distance between two points as,

$$d((x_1, y_1, z_1), (x_2, y_2, z_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{5}$$

As in the two-dimensional case, it is possible to define alternative distance metrics, including the L^1 Manhattan distance.

We may proceed and identify 4-dimensional space with the set of all 4-tuples of numbers. Of course, 4-dimensional space does not render itself to nice drawings and it is hard to get a geometric understanding of it, but the mathematics works similarly to 2D and 3D space. It does not stop at 4 dimensions, we may define n-dimensional space for any positive natural number. In particular, the definition of distance generalizes to n dimensions.

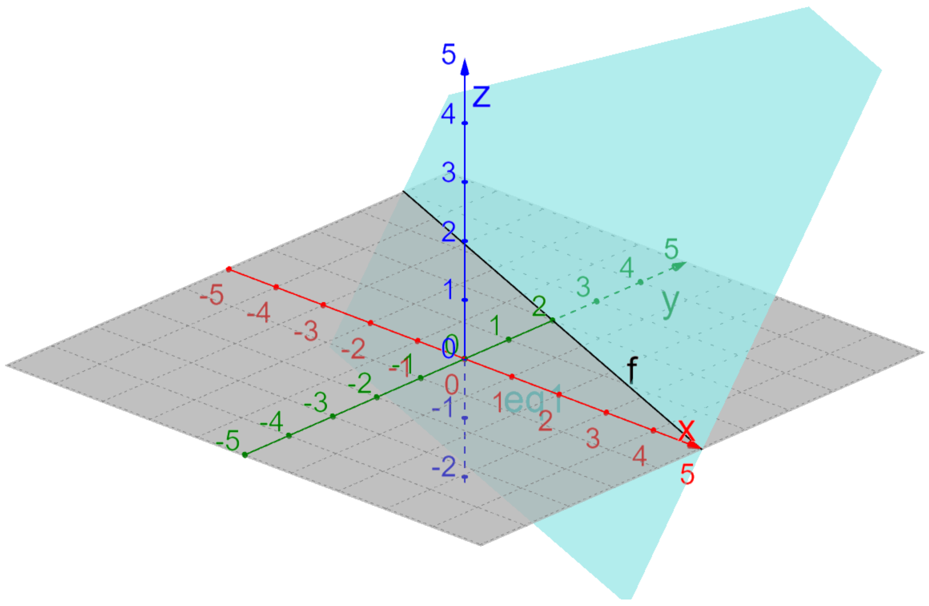

Figure 6: A plane (Made with GeoGebra)

$$d((x_1, x_2, \ldots, x_n), (y_1, y_2, \ldots y_n)) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} \tag{6}$$

for any two numbers $(x_1, x_2, \ldots, x_n)$ and $(y_1, y_2, \ldots y_n)$ in n-space.

1.4 Planes in space

In the plane, the equation (3) defines a straight line. In space, a similar equation defines a plane; i.e., if not $a = b = c = 0$, the points (x, y, z) satisfying

$$ax + by + cz + d = 0 \tag{7}$$

is a plane. Figure 6 illustrates the plane $2x + 5y - 4z - 10 = 0$. Similarly to the 2D case, we may determine the intercepts with the axis. Considering $y_1 = z_1 = 0$, we find the intercept with the x-axis, $x_1 = -\frac{d}{a} = -\frac{-10}{2} = 5$. Similarly, we may find the intercept with the y-axis, 2, and with the z-axis, -2.5. By the way, what do the planes $2x + 5y - 10 = 0$ and $2x - 10 = 0$ look like?

Some observations:

- If $d = 0$, the plane will pass through the origin.
- If $k \neq 0$, then $ax + by + cz + d = 0$ and $kax + kby + kcz + kd = 0$ will describe the same plane.
- If $d \neq e$, $ax + by + cz + d = 0$ and $ax + by + cz + e = 0$ describe parallel planes (i.e., planes with no common points.)
- Two planes that are not parallel, intersect in a straight line.

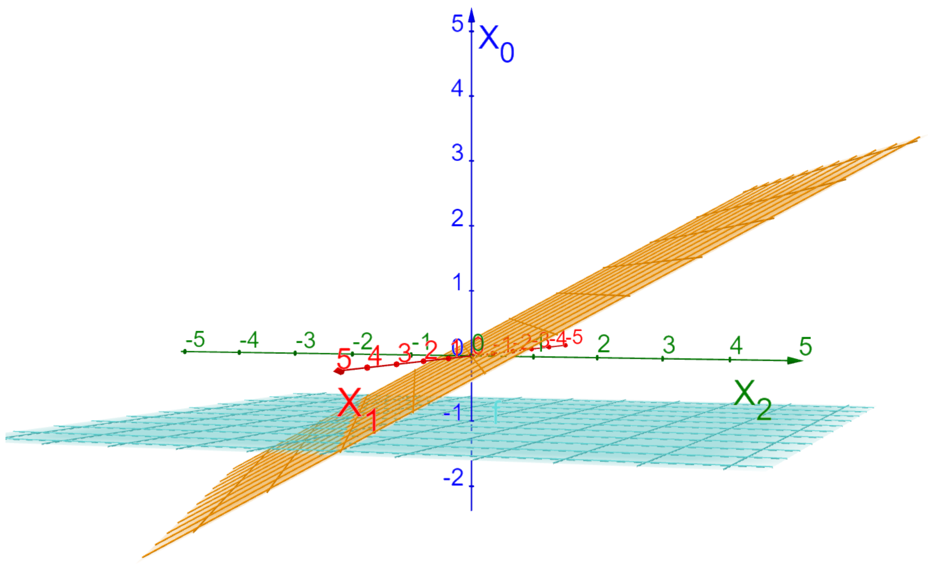

Figure 7: Crossing planes (Made with GeoGebra)

As an illustration of the last point, it should be possible in figure 6 to see how the plane $2x + 5y - 4z - 10 = 0$ intersects the plane $z = 0$ in the straight line $2x + 5y - 10 = 0$.

Figure 7 shows how the plane $2x_1 - 5x_2 + 10x_0$, which passes through the origin, intersects the plane $x_0 = -1$. This illustrates a situtaion in machine learning with two numerical features and a linear classifier (e.g., perceptron, linear regression, or logistic regression). The two numerical features are given by x_1 and x_2. We add a bias feature $x_0 = -1$. An observation will then get the form $(-1, x_1, x_2)$ and will be located in the turqoise plane $x_0 = -1$. The classifier will assign the positive class to the points where $-w_0 + w_1x_1 + w_2x_2 > 0$. The expression $-w_0x_0 + w_1x_1 + w_2x_2 = 0$ describes a plane through the origin, in this case $(-10)x_0 - 2x_1 + 5x_2 = 0$ which intersects the turqoise plane in a straight line $-2x_1 + 5x_2 + 10 = 0$. This corresponds to the decision boundary we normally draw together with the scatterplot in the x_1x_2-plane.

Remark It is not always easy to make good 2D-drawings of 3D-phenomena, and you might find it hard to interpret some of our figures. You might, however, go to GeoGebra.org and make similar figures and play with them yourself. There you may rotate the figures and watch them from different angles.

1.5 Python

We may represent points in n-dimensional space by tuples of length n in python. It is straightforward to define a distance function for these representations which works independently of n. We first present it in a procedural way.

```
def dist_proc(a, b):
    # Euclidean distance in a procedural way
    s = 0
    for (x,y) in zip(a,b):
        s += (x - y) ** 2
    return s ** 0.5
```

This might be similar to the way you are used to implement such functions from earlier courses. In Python, it is possible to write this more compactly using list comprehension.

```
def distance_L2(a, b):
    "L2-distance using comprehension"
    s = sum((x - y) ** 2 for (x,y) in zip(a,b))
    return s ** 0.5
```

We will use list comprehension a lot in this course, and you should make sure you are able to read, understand, and use it.

We wait with the representation of lines and planes in Python until we have introduced some more theory.

2 Vectors

2.1 Vector operations

Vectors and vector spaces are key concept in modern mathematics. These concepts developed gradually from the Cartesian geometry over more than 200 years. Vectors and vector spaces are quite general concepts and there are many different vector spaces. We will mainly be considered with vectors as tuples of real numbers, and we start with the simplest ones, pairs of reals. Pairs of real numbers can be consider as vectors if we equip them with two operations, addition and multiplication with real numbers. In the context of vectors, we call the reals for *scalars*, and multiplication by them for *scalar multiplication.* The two operations are defined componentwise by

Addition: $(x_1, y_1) + (x_2, y_2) = (x_1 + x_2, y_1 + y_2)$

Scalar multiplication: $a(x, y) = (ax, ay)$

We see that this also yields other operations, like subtraction and division by a non-zero scalar, and that there is an identity element $(0, 0)$ for addition:

- $(x_1, y_1) - (x_2, y_2) = (x_1, y_1) + (-1)(x_2, y_2)$
- $(x, y)/a = (1/a)(x, y)$
- $(x, y) + (0, 0) = (x, y)$

We can also observe that the operations are in many respects "well-behaved". Given three vectors $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$, and two scalars c and d, then

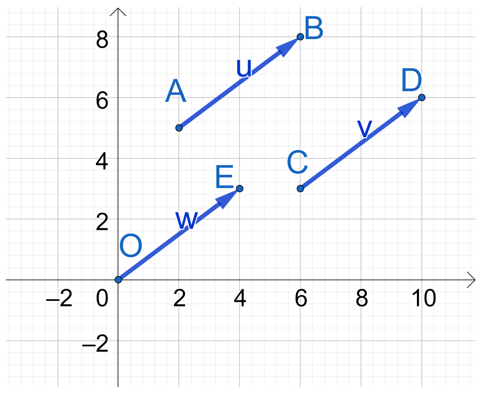

Figure 8: Equivalent vectors (Made with GeoGebra).

- $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$ (Commutativity)
- $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$ (Associativity)
- $c(d\mathbf{u}) = (cd)\mathbf{u}$ (Associativity)
- $c(\mathbf{u} + \mathbf{v}) = c\mathbf{u} + c\mathbf{v}$ (Distributivity)
- $(c + d)\mathbf{u} = c\mathbf{u} + d\mathbf{u}$ (Distributivity)

The set of pairs of real numbers with these operations makes up a vector space. We will call it V_2. We can similarly consider all triples of real numbers together with componentwise addition and multiplication with scalars. They make up a vector space V_3 different from V_2. More generally, for any natural number n, we have a vector spcae consisting of all n-tuples with the componentwise operations.

2.2 Vectors in the plane

We can define vectors as pairs of numbers without bringing in geometry. There is, however, a long tradition of using vectors to describe entities that have a direction and a magnitude, for example forces, velocity, acceleration, to mention a few examples. These vectors can be depicted as arrows in the plane, or more generally, the space, where the magnitude is reflected by the length of the vector. Any ordered pair (A, B) of points in the plane (or space) define an *Euclidean*, or *geometric*, vector with A as starting point and B as end point. The direction from starting point to end point is indicated with an arrowhead.

The important move which links this to vectors as pairs of triples of numbers, is to consider two vectors that have the same direction and length as equivalent. Thus, the three vectors in figure 8 are all equivalent. For all our practical purposes, they can be considered to be the same vector, and can be identified with the vector $\overrightarrow{OE}$ in the figure. We can refer to vectors starting in the origin, $(0, 0)$, by their endpoint, thus to $\overrightarrow{OE}$ by $E = (4, 3)$.

The geometric interpretation of addition and multiplication with scalars may be illustrated as by figure 9. It is only the direction and length of the result that matters.

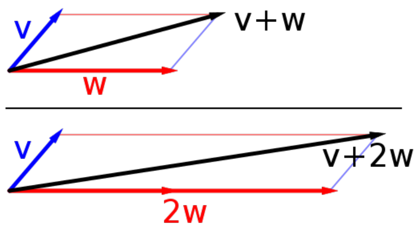

Figure 9: Vector addition and multiplication with scalars (Wikipedia).

A word on notation. Traditionally, one has used an arrow or a bar over the name of the vector to distinguish between vectors and scalars—at least in handwriting. In printing, it has become more usual to use bold face for vectors. Thus $\mathbf{w}$, $\vec{w}$, $\bar{w}$, $\overrightarrow{OE}$ and E are variations of notations for the vector $(4, 3)$ in the figure 8.

Vectors in the plane have a *length*. The length of the vector is also called the *norm* of the vector. For a vector $\mathbf{v}$, the norm is written $||\mathbf{v}||$ in symbols. It should be no surprise that this is the same as the distance between the end-points of the vector. Thus, for the vector $\mathbf{v}$ from (x_1, y_1) to (x_2, y_2), we get

$$\textit{length of } \mathbf{v} = ||\mathbf{v}|| = d((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{8}$$

For a vector $\mathbf{v} = (x, y)$ starting in $(0, 0)$ this reduces to

$$||\mathbf{v}|| = \sqrt{x^2 + y^2} \tag{9}$$

2.3 Cosine

We take one step back and consider angles and trigonometric functions in the plane, in particular the cosine function. In a right-angled triangle, as the one in figure 2, the *cosine* of the angle A is the ratio of the (length of) the adjacent side and the hypotenuse, and the *sine* of the angle A is the ratio of the (length of) the opposite side and the hypotenuse.

$$\cos(A) = \frac{b}{h}, \quad \sin(A) = \frac{a}{h} \tag{10}$$

When $h = 1$, $\cos(A) = b$ and $\sin(A) = a$.

These concepts generalize to obtuse (non-acute) angles by considering the unit circle in the Cartesian plane, cf. figure 10. Since the radius of this circle is 1, the cosine of the angle u ($\angle BOC$) is the x-coordinate of C, i.e., 0.5. This is generalized to the obtuse angle, such that the cosine of the angle $v (= \angle BOD)$ is the x-coordinate of D, which equals -0.9. This makes perfectly sense. The cosine of an angle v

- equals 1 if and only if the angle is 0;
- equals 0 if and only if the angle is a right angle, in radians $\pi/2$;
- is between 0 and 1 if an only if the angle is acute (i.e., between 0 and $\pi/2$);
- is negative if the angle is obtuse, between $\pi/2$ and π.

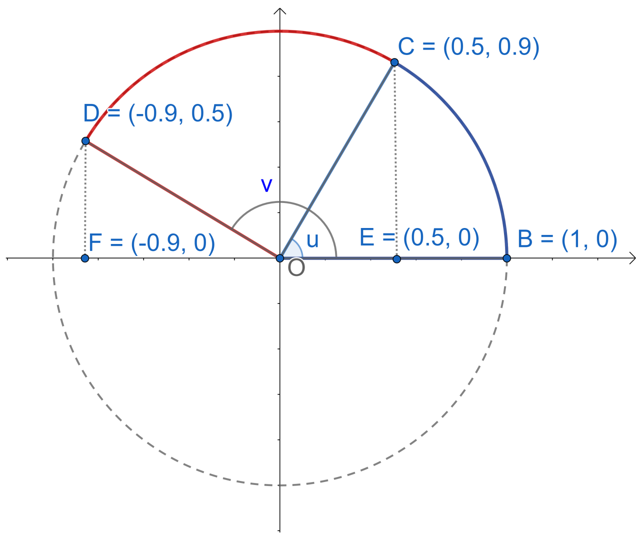

Figure 10: Angles and cosine (Made with GeoGebra)

As you see, we prefer to use radians when talking about the size of an angle—not degrees. The radians of the angle u ($=\angle BOC$) equal the length of the arc BC divided with the radius of the circle, OB. When we use the unit circle, the radius OB has length 1 and the radian of u equals the length of the arc BC. For this particular choice of C, this is $\pi/3$.

2.4 Dot product

We return to the vectors and introduce the important and useful concept of *dot product*, which is also called *inner product*, or *scalar product*. For two vectors in the plane, $\mathbf{a} = (a_1, a_2)$ and $\mathbf{b} = (b_1, b_2)$, it is defined to be

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 \tag{11}$$

Beware that the dot product is <u>not</u> a vector, it is real number, a scalar.

The dot product is related to the cosine according to the following formula for all non-zero vectors $\mathbf{a}$ and $\mathbf{b}$, where v is the angle between the two vectors.

$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}||\ ||\mathbf{b}|| \cos(v) \tag{12}$$

This is equivalent to

$$\cos(v) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||\ ||\mathbf{b}||} \tag{13}$$

We can immediately verify that this is correct for the angles u and v in figure 10, e.g.,

$$\frac{\overrightarrow{OB} \cdot \overrightarrow{OD}}{||\overrightarrow{OB}||\ ||\overrightarrow{OD}||} = (1,0) \cdot (-0.9, 0.5) = -0.9 = \cos(v)$$

But it also holds in general. (Look up a suitable text book for a proof if you are in doubt.) So for example, we get

$$\cos(v-u) = \cos(\angle COD) = \frac{\overrightarrow{OC} \cdot \overrightarrow{OD}}{||\overrightarrow{OC}||\ ||\overrightarrow{OD}||} = (0.5, 0.9) \cdot (-0.9, 0.5) = 0$$

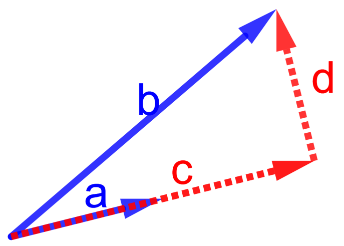

Figure 11: Decomposing a vector (Made with GeoGebra)

(By the way, the numbers are not exact, they are approximations; they have been rounded by GeoGebra. When C's x-coordinate is 0.5, its y-coordinate should have been $\sqrt{1-x^2} = \sqrt{1-0.5^2} = \frac{1}{2}\sqrt{3}$, which has here been rounded off to 0.9.)

A few observations regarding the dot product. First, there is a connection between the dot product and the length of a vector.

$$\mathbf{a} \cdot \mathbf{a} = a_1 a_1 + a_2 a_2 = ||\mathbf{a}||^2 \tag{14}$$

We could have defined the length from the dot product.

Secondly, we observe that for any vector $\mathbf{a}$, the vector $\frac{\mathbf{a}}{||\mathbf{a}||}$ is a vector of length 1 pointing in the same direction as $\mathbf{a}$.

Thirdly, given two vectors $\mathbf{a}$ and $\mathbf{b}$. The vector $\mathbf{b}$ can be written as the sum of two vectors; a vector $\mathbf{c}$, pointing in the same direction as $\mathbf{a}$, and a vector $\mathbf{d}$, orthogonal to $\mathbf{a}$ (see figure 11). We call $\mathbf{c}$ *the projection of* $\mathbf{b}$ *along* $\mathbf{a}$. We can calculate the length of $\mathbf{c}$ to be

$$||\mathbf{c}|| = ||\mathbf{b}|| \cos(\mathbf{v}) = ||\mathbf{b}|| \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}|| \, ||\mathbf{b}||} = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||}$$

and $\mathbf{c}$ to be

$$c = ||\mathbf{c}|| \frac{\mathbf{a}}{||\mathbf{a}||} = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||^2}\mathbf{a} = \frac{\mathbf{a} \cdot \mathbf{b}}{\mathbf{a} \cdot \mathbf{a}}\mathbf{a} \tag{15}$$

If $\mathbf{a}$ has length 1,

$$\mathbf{c} = (\mathbf{a} \cdot \mathbf{b})\mathbf{a} \tag{16}$$

This is the projection of $\mathbf{b}$ in the direction of $\mathbf{a}$.

2.5 Lines and vectors

We will see how lines can be defined in terms of vectors. We will in the following not distinguish between points and vectors (starting in the origin). Consider a vector $\mathbf{u} = A = \overrightarrow{OA} = (a_1, a_2)$ starting in the origin. This determines a straight line where a point (x, y) is on the line if and only if it is of the form $t\mathbf{u} = t(a_1, a_2)$ for some real number t, cf. figure 12.

Vectors yield yet another possibility for characterizing lines. Consider a vector $N = (n_1, n_2)$ orthogonal to the vector $A = (a_1, a_2)$. Since they are orthogonal,

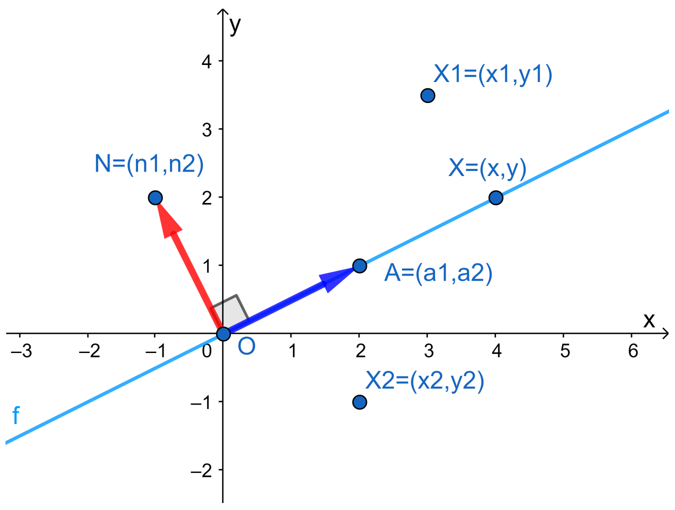

Figure 12: Vector definition of line through the origin (Made with GeoGebra)

$N \cdot A = 0$. Moreover, for any point X on the line through O and A, we have $X = tA$ for some t, hence $X \cdot N = (tA) \cdot N = t(A \cdot N) = 0$. Conversely, we can show that if $X \cdot N = 0$, X has to be on the form tA for some t and be on the line. Hence, a line through the origin is completely determined by a normal vector N to the line. It consists of all points X such that $X \cdot N = 0$.

What if the line does not pass through the origin? It is determined by a vector, $A = (a_1, a_2)$ and a point on the line $B = (b_1, b_2)$. A point (x, y) is on the line if and only if it is of the form

$$B + tA = (b_1, b_2) + t(a_1, a_2) \tag{17}$$

for some real t.

Also the line not passing through the origin can be characterized by a normal vector, N, together with one point on the line, say B. A point $X \neq B$ will be on the line if and only if the vector $\overrightarrow{BX}$ is orthogonal to N; hence if and only if $(X - B) \cdot N = 0$.

In other words, a point X will be on the line if and only if $X \cdot N = B \cdot N$. The real number $X \cdot N$ will be the same, say s, for all numbers X on the line.

For points not on the line, we use the orientation of the normal vector N to talk about points above or below the line. For a point above the line, e.g. point P in figure 12, $P \cdot N > s$, while for a point below the line, $Q \cdot N < s$.

Observe the importance of the orientation of the normal vector N. We could equally well have used the normal vector $N_2 = -N$ to define the same line. Then a point X would be on the line if and only if $X \cdot N_2 = -s$. But now $Q \cdot N_2 > B \cdot N_2$, while $Q \cdot N < B \cdot N$.

Linear classifiers in machine learning can be intepreted in this picture. There an observation $\mathbf{x}$ belongs to the positive class if $\mathbf{w} \cdot \mathbf{x} > \theta$ for some threshold θ. (If we include a bias in $\mathbf{x}$, we can assume $\theta = 0$.) The observation $\mathbf{x}$ is the same as the point P, while the normal vector N corresponds to the weight vector $\mathbf{w}$.

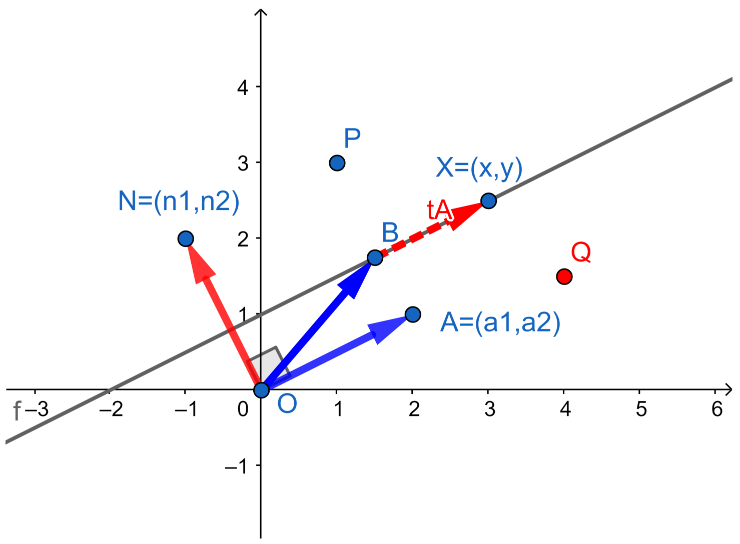

Figure 13: Line not through the origin. (Made with GeoGebra)

2.6 Vectors in space; lines and planes

Three-dimensional vectors have a geometric interpretation in 3D-space similar to the interpretation of two-dimensional vectors in 2D-space. Much of what we saw for 2D-vectors in the plane carry over to vectors in space. Dot product is defined similarly. For two vectors in the space, $\mathbf{a} = (a_1, a_2, a_3)$ and $\mathbf{b} = (b_1, b_2, b_3)$, it is defined to be

$$\mathbf{a} \cdot \mathbf{b} = a_1 b_1 + a_2 b_2 + a_3 b_3 \tag{18}$$

The relationships between dot product and length and between dot product and cosine, cf. equations 12 and 13, also hold in space.

A line in space can be characterized by the same vector equation 17 as in the plane. A normal vector does not suffice to determine a line uniquely, however.

A plane in space can be defined similarly to the definition of a line by equation 17 by a point C and two vectors A and B that are not parallel. Two vectors A and B are said to be parallel if $B = tA$ for some real t.

$$C + sA + tB = (c_1, c_2) + s(a_1, a_2) + t(b_1, b_2) \text{ for any } t \text{ and } s. \tag{19}$$

A plane is uniquely determined by three points in the plane not on a straight line. Given three points P, Q, R, we get the plane by setting $C = P$, $A = \overrightarrow{PQ}$ and $B = \overrightarrow{PR}$ in equation 19.

Let N be a vector which is orthogonal to A and to B. For any X in the plane defined by formula 19, the following holds.

$$(X - C) \cdot N = (C + sA + tB - C) \cdot N = sA \cdot N + tB \cdot N = 0 \tag{20}$$

Conversely, any X satisfying equation 20 will be in the plane defined by equation 19. We will call N a *normal vector to the plane.* We see that the plane can be defined by the normal vector similarly to the way the line in the 2D plane could be defined by a normal vector.

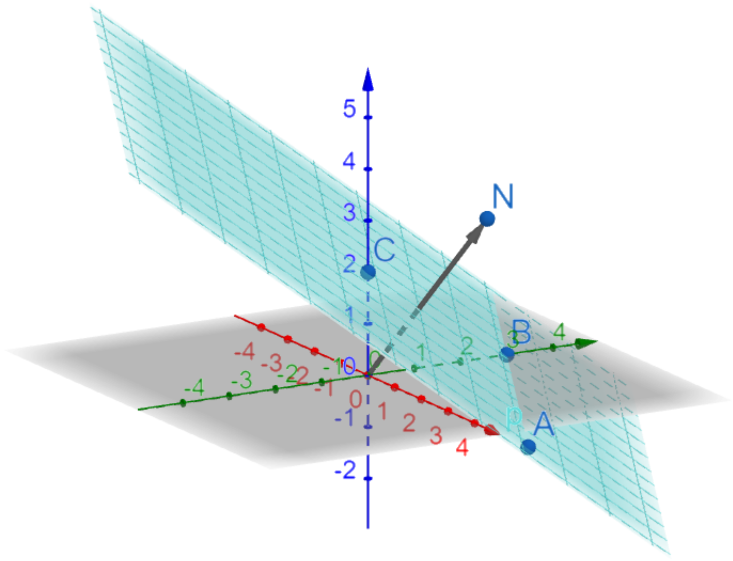

Figure 14: A plane with a normal vector. (Made with GeoGebra)

We also see that for all X in the plane $X \cdot N = C \cdot N$. Call this e.g. d. A point X will be above the plane, defined by the direction of N, if and only if $X \cdot N > d$ and below the plane if and only if $X \cdot N < d$.

Say that N has the coordinates $N = (a, b, c)$. Then $N \cdot X = ax + by + cz$ and we recognize that $X \cdot N = d$ gets the form $ax + by + cz = d$, similarly to the equation for a plane in equation 7.

The relationship to machine learning is similar to the 2D case. The vector N corresponds to the weight vector w, and the plane is the decision border for the classifier given these weights.

2.7 Higher dimensions

It is not easy to imagine what spaces of higher dimensions than three look like. But many of the concepts from two and three dimensional space can be generalized to these higher dimensional spaces. First, dot product is straightforwardly generalized by

$$(a_1, a_2, \dots a_n) \cdot (b_1, b_2, \dots c_n) = a_1b_1 + a_2b_2 + \cdots + a_nb_n \tag{21}$$

Even though it is hard to imagine vectors and angles in these spaces, we can define the angles between vectors from the dot product by formula 13, repeated here.

$$\cos(v) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}||\ ||\mathbf{b}||} \tag{22}$$

It can be shown that with this definition many of the properties of angles from two and three dimensional spaces carry over. The cosine will always take values between -1 and 1, and it makes sense to say that two vectors are orthogonal if and only if their dot product equals zero.

The concept of a line in 2D and a plane in 3D can be generalized to a *hyperplane*. Given a vector $N = (a_1, a_2, \dots a_n)$. All points $X = (x_1, x_2, \dots x_n)$ such that

$$N \cdot X = (a_1, a_2, \dots a_n) \cdot (x_1, x_2, \dots x_n) = 0 \tag{23}$$

defines a hyperplane through the origin with N as normal vector.

Given a point C such that $N \cdot C = d$. Then all points $X = (x_1, x_2, \dots x_n)$ such that

$$N \cdot X = (a_1, a_2, \dots a_n) \cdot (x_1, x_2, \dots x_n) = d \tag{24}$$

defines a hyperplane through C with N as normal vector. As before, a point X will be above the plane if and only if $X \cdot N > d$ and below the plane if and only $X \cdot N < d$. The hyperplane will separate the n-dimensional space into two halves, the points above the hyperplane and the points on or below the hyperplane.

2.8 Detour: General vector spaces

The set of pairs of real numbers with the two operations of componentwise addition and scalar multiplicationis an example of a *vector space*. The set of triples of real numbers with addition and scalar multiplication defined componentwise makes another vector space. For any natural number n, the set of n-tuples with the componentwise operations makes a vector space.

In mathematics, a *vector space*, or *linear space*, is any mathematical structure satisfying a certain set of axioms. First, one has to determine the scalars. These can be the reals, but it can alternatively be the complex numbers or rationals or any structure which in mathematics is called a *field*. Then, one has to define the two operations of addition and scalar multiplication such that they satisfy a set of axioms.

For example, the set of all continuous function from real numbers to real numbers makes a vector space, where addition and scalar multiplication is defined as

$$(f + g)(x) = f(x) + g(x)$$

$$(cf)(x) = cf(x)$$

In this course, we will mainly consider vector spaces of n-tuples of reals.

2.9 NumPy

NumPy is a Python package for efficient numerical computation. It has arrays as basic data structures and is convenient for representing vectors. We will not give a tutorial on NumPy here; there exist many. But we will point out how vectors can be represented.

A Python list or tuple can be converted into an NumPy array, which can be considered as a vector.

```
In [1]: import numpy as np
In [2]: a = np.array([1,2,3])
In [3]: a
Out[3]: array([1, 2, 3])
```

```
In [4]: b = np.array((4.5, 6, 7))
In [5]: b
Out[5]: array([4.5, 6. , 7. ])
```

Observe that all entries in an array has to be of the same type. In the definition of b, we mixed integers and floats, but NumPy casted the integers to floats in the construction of b.

Vector addition can be represented by '+' and multiplication by scalars by '*'.

```
In [6]: a+b
Out[6]: array([ 5.5,  8. , 10. ])
In [7]: c = 5.0
In [8]: c*a
Out[8]: array([ 5., 10., 15.])
```

This is interpreted very differently to what happens if we use the same symbols between lists, e.g.

```
In [9]: [1,2,3]+[4.5,6,7]
Out[9]: [1, 2, 3, 4.5, 6, 7]
```

These NumPy operations look like they are taylor-made for vectors, but they are, in fact, examples of more general NumPy mechanisms of componentwise operations. For example, one may multiply arrays componentwise or add a scalar to a vector through these operations do not correspond to vector operations.

```
In [10]: a*b
Out[10]: array([ 4.5, 12. , 21. ])
In [11]: a+c
Out[11]: array([6., 7., 8.])
```

The latter is an example of what we call *broadcasting*. It can be considered as follows. When $c = 5.0$ is added to $a = np.array([1, 2, 3])$ then c is first transformed to the array $c' = np.array([c, c, c])$ to get the same dimension as a, and then the array c' is added to a.

One of the great operational savings one gets by using numpy comes from the possibility to apply a function to an array resulting in a componentwise application, e.g,

```
In [12]: np.cos(a)
Out[12]: array([ 0.54030231, -0.41614684, -0.9899925 ])
```

In other words, if $a = np.array([a_1, a_2, a_3])$ then

$$np.cos(a) = np.array(np.cos(a_1), np.cos(a_2), np.cos(a_3)).$$

Observe that $a * b$ is not the dot product. NumPy contains a function for calculating the dot product, which may be written two different ways.

```
In [13]: np.dot(a,b)
Out[13]: 37.5
In [14]: a.dot(b)
Out[14]: 37.5
```

Newer version of Python/NumPy also has a possibility for expressing this with the infix operator @. This makes longer expressions more readable and compact.

```
In [15]: a @ b
Out[15]: 37.5
```

For more information on NumPy we recommend

- the documentation `https://numpy.org/doc/stable/`,
- and, in particular, the introduction `https://scipy-lectures.org/` which also presents other of the tools we will use, like plotting and scikit-learn.

3 Matrices

3.1 Basic definitions and notation

A matrix is a rectangular array of scalars. It is normally enclosed in square brackets, as in the following example

$$B = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix} \tag{25}$$

Sometimes—particularly in hand-writing—ordinary parentheses are used instead of square brackets as in the following example

$$C = \begin{pmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{pmatrix} \tag{26}$$

B has 2 *rows* and 3 *columns*; and its dimension is said to be 2×3, which is read *two by three.* Matrix C has dimension 3×3.

A scalar may be multiplied with a matrix componentwise to get a matrix of the same dimension, e.g.,

$$5B = \begin{bmatrix} 55 & 60 & 65 \\ 105 & 110 & 115 \end{bmatrix} \tag{27}$$

Two matrices with the same dimensions can also be added componentwise, e.g.

$$B + \begin{bmatrix} 11 & 22 & 33 \\ 21 & 22 & 23 \end{bmatrix} = \begin{bmatrix} 22 & 34 & 46 \\ 42 & 44 & 46 \end{bmatrix} \tag{28}$$

Thus, the set of matrices over a given set of scalars with the same dimensions can be considered a vector space.

Some words on notation. It is not a strict rule, but it is usual to use capital roman letters to denote matrices (and to use bold face lowercase letters for vectors). Some, e.g., Marsland, in addition choose to boldface the matrices. We refer to the elements of the matrix by two indices, one for row and for column as in the following example for an $m \times n$ matrix.

$$X = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \cdots & x_{1,n} \\ x_{2,1} & x_{2,2} & x_{2,3} & \cdots & x_{2,n} \\ x_{3,1} & x_{3,2} & x_{3,3} & \cdots & x_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & x_{m,3} & \cdots & x_{m,n} \end{bmatrix} \tag{29}$$

Thus $x_{3,2}$ is the element in the third row and second column, and, more generally, $x_{i,j}$ is the element in the ith row and jth column. It will sometimes be convenient to refer to this element also by $X_{i,j}$, in particular when we consider the product of two matrices, e.g., to refer to element i, j of AB, we use $(AB)_{i,j}$. Yet another way of referring to this element, is to refer to it by $X[i, j]$. We will see that this can come in handy if we want to refer to whole rows or columns of matrices. More on that below.

In mathematics, it is usual to start counting at 1, while in programming, for example Python, the counting starts at 0 and the indices in the first row will run from $x_{0,0}$ to $x_{0,n-1}$.

3.2 The transposed of a matrix

By interchanging rows and columns in a matrix A, we get the *transposed* of A, in symbols A^T. For example, the transposed of the matrix B in equation 25 is the following.

$$B^T = \begin{bmatrix} 11 & 21 \\ 12 & 22 \\ 13 & 23 \end{bmatrix} \tag{30}$$

A general definition of the transposed is that for a $m \times n$ matrix A, the transposed of A is the matrix A^T given by $(A^T)_{j,i} = A_{i,j}$ for all i, j such that $1 \leq i \leq m, 1 \leq j \leq n$.

3.3 Matrix multiplication

When A is an $m \times n$ matrix and B is an $n \times p$ matrix, we can define the product $C = AB$. This is a $m \times p$ matrix. It is defined by the following definition for each entry

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} \text{ for all } i, j \text{such that} 1 \leq i \leq m, 1 \leq j \leq p \tag{31}$$

This is illustrated by figure 15, where e.g., the second entry in the first row of AB is $a_{1,1}b_{1,2} + a_{1,2}b_{2,2}$. We see that we multiply the first element in the first row of A with the first element in the second column of B, then we multiply together the second elements from the two, and so on, before we finally sum these products together. This can alternatively be written as a dot product of two vectors $(a_{1,1}, a_{1,2}) \cdot (b_{1,2}, b_{2,2})$. In general,

$$c_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j} = (a_{i,1}, a_{i,2}, \ldots, a_{i,n}) \cdot (b_{1,j}, b_{2,j}, \ldots, b_{n,j}) \tag{32}$$

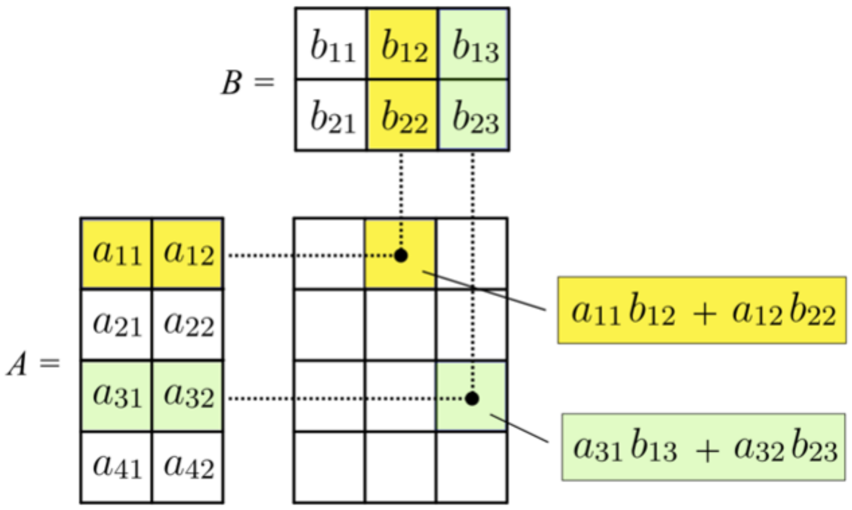

Figure 15: Matrix multiplication (Wikipedia)

Let us use the notation $X[i,:]$ to denote the vector consisting of the elements of row i of X, and correspondingly, use $X[:,j]$ to denote the vector consisting of the elements of column j of X. This means that

$$X[i,:] = (x_{i,1}, x_{i,2}, \dots, x_{i,n})$$

$$X[:,j] = (x_{1,j}, x_{j,2}, \dots, x_{j,m})$$

in the matrix of equation 29. Observe that with this notation, $X[i,:]$ and $X[:,j]$ are vectors, while $X[i,j]$ is a scalar, i.e., a real number. Finally, $X[:,:] = X$, the matrix itself. Other books use other notation, e.g., $*$ or $\cdot$ where we have used the colon.

In this notation we can rewrite the formula for matrix multiplication as

$$(AB)[i,j] = A[i,:] \cdot B[:,j] \text{ for all } i,j \text{ such that} 1 \leq i \leq m, 1 \leq j \leq p \qquad (33)$$

Observe that AB does not have to equal BA. The product BA is not even defined unless also $m = p$. Moreover, if A and B have different dimensions, say 2×3 and 3×2, resp., AB will have dimension 3×3 while BA will have dimension 2×2. Even when A and B are both square, say $n \times n$ matrices, AB does not have to equal BA.

Multiplication of matrices have, however, some of the properties one could expect from multiplication. In particular, when the expressions are meaningful

$$A(BC) = (AB)C \text{ (associativity)} \qquad (34)$$

A word of warning. Matrix multiplication generalizes the dot product in the sense that each entry in the product matrix can be considered the dot product of rows and columns in the factor matrices. But the matrix multiplication itself is not a dot product. The dot product is a scalar. The matrix product is a new (two-dimensional) matrix. In some machine learning texts, one may see the notation $A \cdot B$ for the matrix multiplication. We think this is misleading and should be avoided.

There is a straightforward relationship between matrix multiplication and the transposed, which is wise to remember.

$$(AB)^T = B^T A^T \qquad (35)$$

3.4 Row and column vectors

A column vector is an $n \times 1$ matrix, e.g.

$$X = \begin{bmatrix} x_{1,1} \\ x_{2,1} \\ \vdots \\ x_{n,1} \end{bmatrix} \tag{36}$$

A row vector is a $1 \times n$ matrix, e.g

$$Y = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \end{bmatrix} \tag{37}$$

Column vectors can be used for representing vectors, e.g., X in equation 36 can be used for representing $x[1,:] = \mathbf{x} = (x_{1,1}, x_{2,1}, \ldots x_{n,1})$. This can simplify certain operations, as they can be described matrix multiplication.

Let us see how this can be applied in machine learning. We have a set of data-points, say $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$. Each of them is represented by an m-dimensional vector, thus $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \ldots x_{i,m})$. We also have a weight vector $\mathbf{x} = (w_1, x_2, \ldots x_m)$. Consider an ML algorithm where we are to compute a forward step for all the input points, i.e., we will calculate $y_i = \mathbf{x}_i \cdot \mathbf{w}$ for each $\mathbf{x}_i$. We can represent the data set as a $N \times m$ matrix, X, where the row i represents the vector $\mathbf{x}_i$, i.e., $X[i,:] = \mathbf{x}_i$. The weight vector, $\mathbf{w}$, can be represented as a column vector W, i.e., $W[:,1] = \mathbf{w}$. This yields the following set-up (where we have renamed each component of $\mathbf{w}$ by $w_{1,j} = w_j$).

$$\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix} \begin{bmatrix} w_{1,1} \\ w_{2,1} \\ \vdots \\ w_{m,1} \end{bmatrix} = \begin{bmatrix} y_{1,1} \\ y_{2,1} \\ \vdots \\ y_{N,1} \end{bmatrix} \sim \begin{bmatrix} t_{1,1} \\ t_{2,1} \\ \vdots \\ t_{N,1} \end{bmatrix}$$

We observe that we refind the dot products $y_i = \mathbf{x}_i \cdot \mathbf{w}$ as the corresponding entries $Y[i,1]$ in $Y = XW$.

This becomes even more pronounced when the output for each input vector of length m is an output vector of length n and when there is a weight vector associated with each output dimension. In this case, each weight vector will be represented by a column in an $m \times n$ matrix, and the output matrix will come out as an $N \times n$ matrix where row i will represent the corresponding output vector $\mathbf{y}_i$. This will hopefully become clearer when we come to algorithms where it gets used.

$$\begin{bmatrix} x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{N,0} & x_{N,1} & x_{N,2} & \cdots & x_{N,m} \end{bmatrix} \begin{bmatrix} w_{0,1} & w_{0,2} & \cdots & w_{0,n} \\ w_{1,1} & w_{1,2} & \cdots & w_{1,n} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,1} & w_{m,2} & \cdots & w_{m,n} \end{bmatrix} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N,1} & y_{N,2} & \cdots & y_{N,n} \end{bmatrix}$$

As humans, we can move easily between the vector and the column (or row) vector representing it. When we are to implement vectors and matrices, we have to be more careful, though. There is a difference between the n-dimensional vector, which is a one-dimensional array of n numbers, and the column (or row) vector, which is a two-dimensional array of numbers. When implementing matrices in Python or in NumPy, we have to be fully aware of the difference, as we soon will see.

You may now proceed to the NumPy part if you like. The next two paragraphs will enhance your understanding of matrices and what they are used for. But it is not strictly necessary for applying matrices in NumPy and machine learning.

3.5 Linear transformations and matrices

Matrices serve many purposes in mathematics and will be important in our understanding of machine learning. A basis for understanding them is to see how they are used to represent linear mappings. The notation

$$T : V \to W \tag{38}$$

is used to indicate that T is a mapping of elements in V to elements in W; for $v \in V$, $T(v) \in W$. We will be interested in the case where V and W are vector spaces with the same scalars. The mapping T is then called *linear* provided

- $T(\mathbf{u} + \mathbf{v}) = T(\mathbf{u}) + T(\mathbf{v})$
- $T(c\mathbf{u}) = cT(\mathbf{u})$

for all vectors $\mathbf{u}, \mathbf{v}$ in V and scalars c.

We will assume that V has dimension n and W has dimension m. To ease the discussion, we will let $n = 3$ and $m = 2$, but it should be easy to see how this carries over to any finite n and m. Call the basis elements in V for $\mathbf{e}_1 = (1, 0, 0)$, $\mathbf{e}_2 = (0, 1, 0)$, and $\mathbf{e}_3 = (0, 0, 1)$, respectively. A general element in V has the form $(x_1, x_2, x_3) = x_1\mathbf{e}_1 + x_2\mathbf{e}_2 + x_3\mathbf{e}_3$. If T is a linear mapping, it follows that $T((x_1, x_2, x_3)) = x_1T(\mathbf{e}_1) + x_2T(\mathbf{e}_2) + x_3T(\mathbf{e}_3)$. In other words, the value of T for any element in V is totally determined by the value of T for the three basis elements.

Call the basis elements in W for $\mathbf{f}_1 = (1, 0)$ and $\mathbf{f}_2 = (0, 1)$. Then $T(\mathbf{e}_1) = a\mathbf{f}_1 + b\mathbf{f}_2$ for some a and b. Let us write $a_{1,1}$ for a and $a_{2,1}$ for b, i.e., $T(\mathbf{e}_1) = a_{1,1}\mathbf{f}_1 + a_{2,1}\mathbf{f}_2$, and similarly $T(\mathbf{e}_2) = a_{1,2}\mathbf{f}_1 + a_{2,2}\mathbf{f}_2$ and $T(\mathbf{e}_3) = a_{1,3}\mathbf{f}_1 + a_{2,3}\mathbf{f}_2$.

A general element x_1, x_2, x_3 in V is then mapped to

$$\begin{aligned}
&T(x_1, x_2, x_3)\\
&\quad= \quad x_1(a_{1,1}\mathbf{f}_1 + a_{2,1}\mathbf{f}_2) + x_2(a_{1,2}\mathbf{f}_1 + a_{2,2}\mathbf{f}_2) + x_3(a_{1,3}\mathbf{f}_1 + a_{2,3}\mathbf{f}_2)\\
&\quad= \quad (x_1a_{1,1} + x_2a_{1,2} + x_3a_{1,3})\mathbf{f}_1 + (x_1a_{2,1} + x_2a_{2,2} + x_3a_{2,3})\mathbf{f}_2)
\end{aligned}$$

We see that this corresponds to the matrix

$$m(T) = A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \end{bmatrix} \tag{39}$$

Each column in A tells what the corresponding basis element in V should be mapped to, and each row in A tells what will end up with the corresponding basis element in W.

Any linear transformation T corresponds to such a matrix. Conversely, any $m \times n$ matrix A defines a linear mapping T_A from V_n to V_m where

$$(T_A((x_1, x_2, \ldots, x_n)))_i = (x_1a_{i,1} + x_2a_{i,2} + \cdots + x_na_{i,n}) = \sum_{j=1}^{n} x_ja_{i,j} \tag{40}$$

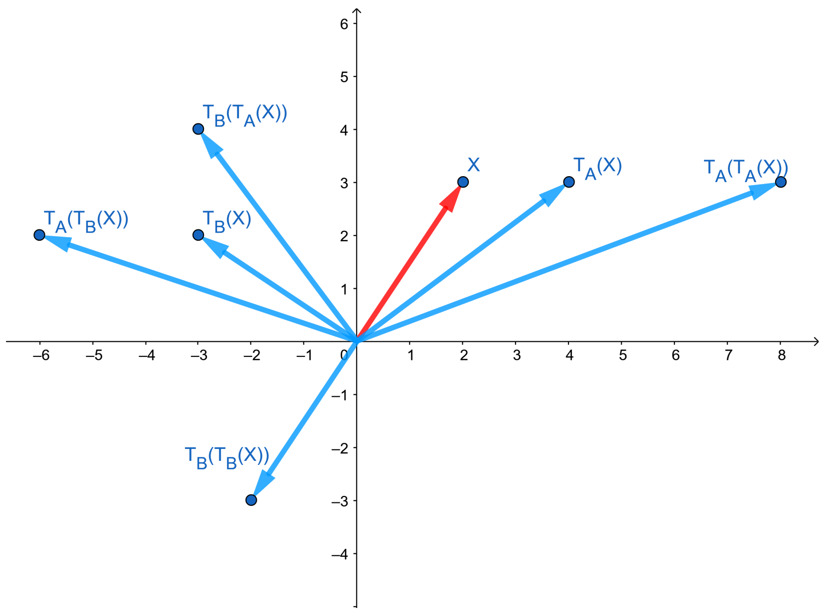

Figure 16: Linear mappimgs (Made with GeoGebra)

3.6 Composition of mappings and multiplication of matrices

To get a better grip of what is going on, let us consider some examples. We will consider vectors in the plane because they are easy to visualize, i.e., we will consider transformations from V_2 to V_2, and start with two matrices.

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{41}$$

Each of them represents a mapping from V_2 to V_2, call them T_A and T_B, resp. What do these mappings do to the vectors? We see that $T_A((x,y)) = (2x,y)$ it doubles the x-value, but leaves the y-value unaltered. While $T_B((x,y)) = (-y,x)$. This corresponds to rotating the vector $\pi/2$ counterclockwise. These examples illustrate some of the transformations one can do by linear mappings.

We can repeat these mappings. For example $T_A(T_A((x,y))) = (4x,y)$ and $T_B(T_B((x,y)) = (-x,-y)$. The latter corresponds to a counterclockwise rotation of π. What happens if we combine the two mappings? We see that $T_B(T_A((x,y)) = (-y,2x)$ and $T_A(T_B((x,y)) = (-2y,x)$. This is all illustrated in figure 16

In general, let U, V, and W be vector spaces and $T : U \to V$ and $S : V \to W$. Write ST for the mapping from U to W defined by $ST(\mathbf{u}) = S(T(\mathbf{u}))$. The example illustrates two important properties of ST. Firstly, when S and T are linear maps, then so is ST. Secondly, in general, ST is not the same as TS.

Since S, T and ST are linear mappings, applying them corresponds to matrix multiplication. Let $m(S)$, $m(T)$ and $m(ST)$ be the corresponding marices. By inspecting the definitions one may prove that $m(ST) = m(S)m(T)$, where we by $m(S)m(T)$ means the matrix multiplication of the two matrices $m(ST) = m(S)m(T)$. In other words, composition of mappings corresponds to the matrix multiplication of the corersponding matrices.

3.7 Matrices in NumPy

NumPy represents matrices similarly to vectors. We can for example transform a list of lists to a matrix.

```
In [3]: a = np.array([[11,12,13],[21,22,23]])

In [4]: a
Out[4]:
array([[11, 12, 13],
       [21, 22, 23]])
```

We can read out that this is a 2×3 matrix by the *shape* attribute.

```
In [5]: a.shape
Out[5]: (2, 3)
```

There is a simple attribute for finding the transposed of a matrix

```
In [6]: a.T
Out[6]:
array([[11, 21],
       [12, 22],
       [13, 23]])
```

The *reshape* attribute makes it possible to change the shape of a matrix. In particular, we can reshape a vector into an $m \times n$ matrix.

```
In [7]: c=np.arange(12)

In [8]: c
Out[8]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

In [9]: d=c.reshape(3,4)

In [10]: d
Out[10]:
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

Matrix multiplication can be expressed by *np.dot*, or @, similarly to the dot product between vectors. To use the same symbol for dot product and matrix multiplication is a somewhat strange design choice, as a dot product between vectors is a scalar, while the product of two matrices is a new matrix. It works fine, but one should always keep in mind the dimension/shape of the objects one considers.

```
In [11]: np.dot(a,d)
Out[11]:
array([[152, 188, 224, 260],
       [272, 338, 404, 470]])
```

```
In [12]: a @ d
Out[12]:
array([[152, 188, 224, 260],
       [272, 338, 404, 470]])
```

Returning to the *reshape*, a convenient trick when we do not know the exact number of entries in a matrix, is to use -1 for the last dimension. NumPy will then choose the dimension that harmonize with the other dimensions.

```
In [13]: c.reshape(4,-1)
Out[13]:
array([[ 0,  1,  2],
       [ 3,  4,  5],
       [ 6,  7,  8],
       [ 9, 10, 11]])
```

In particular, this is useful if we want to transfer a vector to a column (or row vector).

```
In [14]: col = c.reshape(-1,1)

In [15]: col
Out[15]:
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11]])

In [16]: col.shape
Out[16]: (12, 1)

In [17]: c.shape
Out[17]: (12,)
```

Finally, the following example shows how we can add a column of -1-s to the front of a matrix, corresponding to adding a bias feature of -1.

```
In [18]: m = a.shape[0]
In [19]: bias = - np.ones((m,1)) # Makes a m*1 matrix of (-1)s

In [20]: new = np.concatenate([bias, a], axis=1)

In [21]: new
Out[21]:
array([[-1., 11., 12., 13.],
       [-1., 21., 22., 23.]])
```