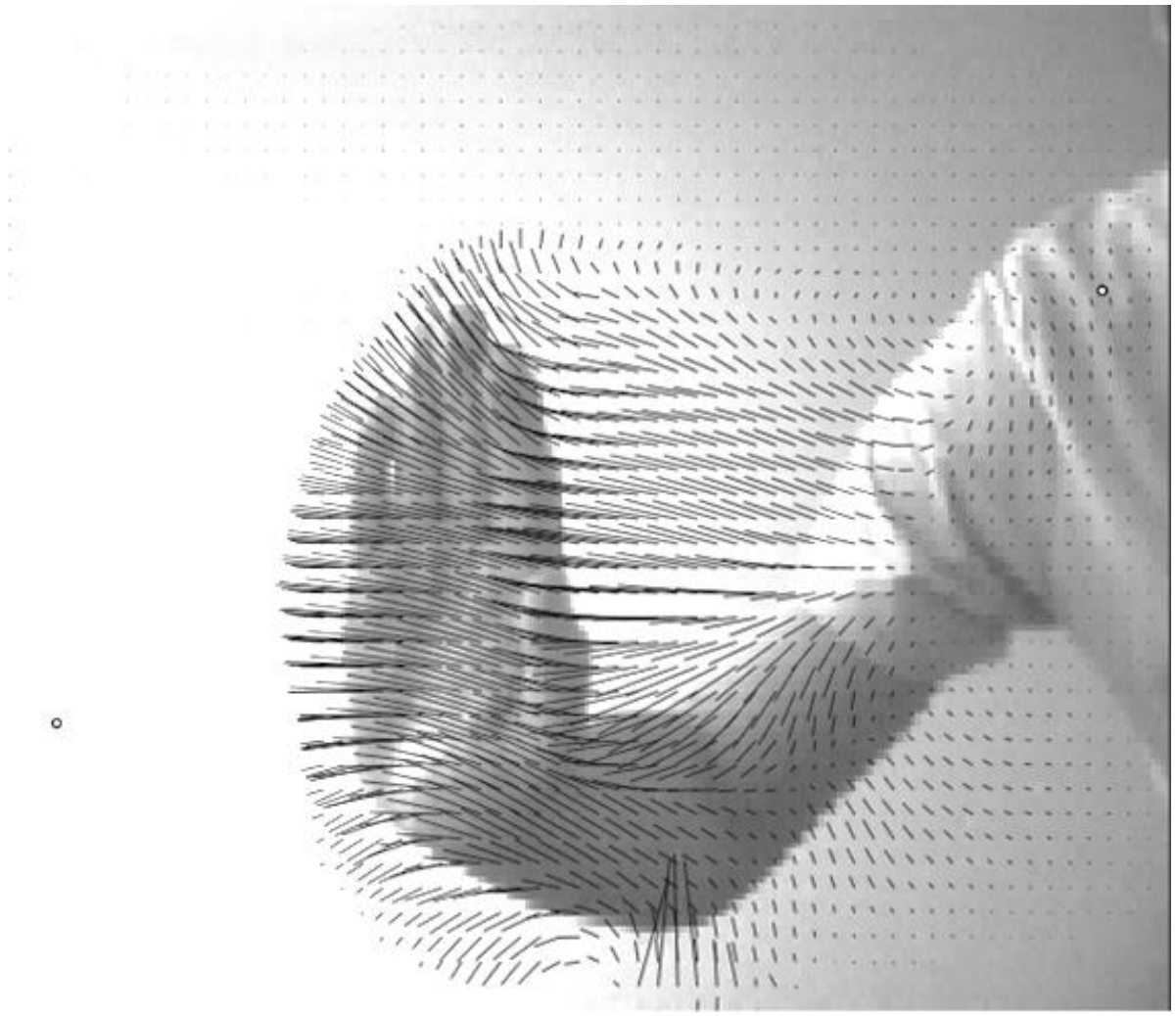
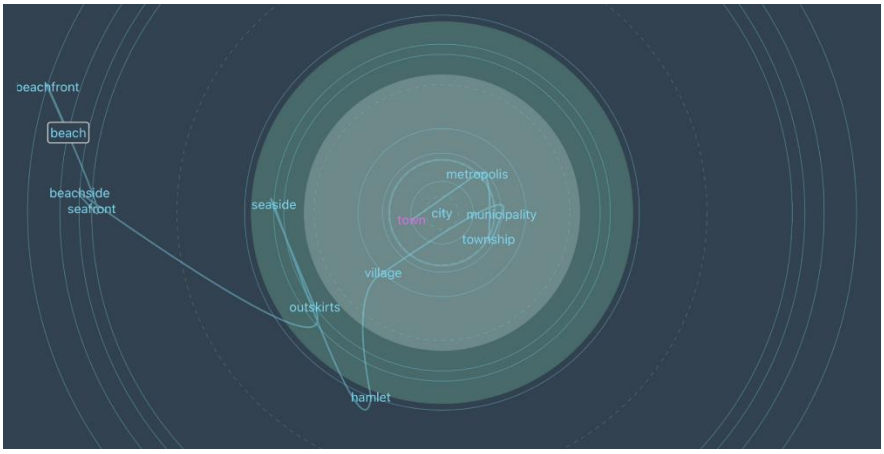


Vector Spaces

Panagiotis Michalatos

Introduction

Machine Learning models often construct or imply some kind of abstract space within which data (images, sounds, text) exist. Objects are embedded in such spaces so that their relative positions are meaningful. For example, in language embeddings related words may appear close to each other, or in image classifications images that contain similar objects cluster together. Navigating one of these spaces is what some generative algorithms are doing when trying to find an artifact (image/audio) that has specific properties that map to specific positions in this abstract space.



These spatial structures can be very rich and informative and there are many numerical tools that enable us to analyse, visualize and traverse them.

In a sense one could argue that when looking at the world through an ML model everything becomes a space of some sort.

In this seminar we will try to first develop intuition about the relevant concepts and techniques when dealing with such abstract spaces of data. To do so we will start with a review of analytical geometry in three-dimensional Euclidean space, the mathematical objects (points and vectors) and operations (distance, dot product) that enable us to describe objects and relations within this space. We will proceed to show how the same concepts transfer to different types of low dimensional spaces, some geometric like the surface of a sphere, or abstract like the space of colours. Finally, we will hint at the extension of these methods to spaces of arbitrary numbers of dimensions like the space of all possible images or the space of words in a language model.

Spaces

We all possess some intuitive understanding of what it means to exist and move within a space of three dimensions, how to establish our locations relative to reference points, measure distances and determine orientation.

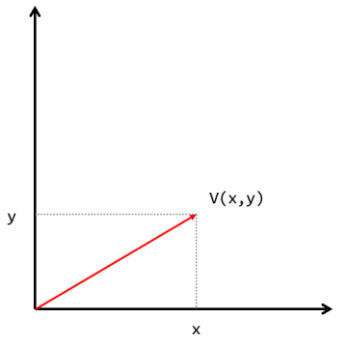
We say that the space we occupy is three dimensional because for example at any point we can define exactly three mutually orthogonal directions of movement. We also have an intuition regarding spaces of lower dimensions. We move on the surface of the earth, and we can use just two numbers, longitude and latitude to pinpoint a location. Although the earth is a three-dimensional object, its idealized surface only has two dimensions and therefore we can map it on a flat piece of paper, creating a map.

When we want to describe geometric objects or relationships in such spaces in a precise manner, whether we are doing physics, architecture or computer graphics, we need to use mathematical objects and operations that depending on the complexity of the problem will fall under the fields of analytical geometry, linear algebra, differential geometry or topology.

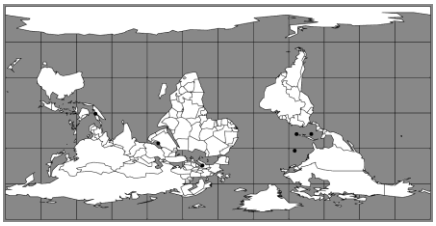
Points

A point is in a sense the minimal building block of space, it represents a location in a space.

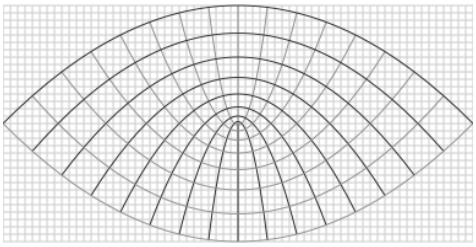
When we want to do calculations involving points, we need some numerical way to define them. The most common convention is to use cartesian coordinates where we define three arbitrary orthogonal directions in space that we call the X, Y and Z axes along with another location that we designate as the origin. Then each point in space can be represented by a unique triplet of numbers {x,y,z} that measure the distance from the origin along each axis of the corresponding orthogonal projection of the point. We call the three numbers {x,y,z} the coordinates of the point.



There are many other coordinates systems (Ways of assigning numerical addresses to points in a space) that are used depending on context.



{Latitude, Longitude} pairs for geographic locations which are a special case of spherical coordinates



Parabolic coordinates

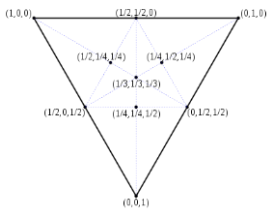
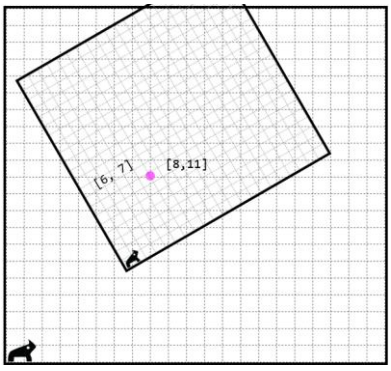


image src: Wikipedia

Or barycentric coordinates over triangles extensively used in computer graphics when one interpolates surface properties (normals, colour, texture) over three dimensional meshes.

It's important to note that a point exists in a space regardless of how we measure it. We can use multiple coordinate systems to describe the same location. For example, we could define a location in the classroom by measuring its projected

distance along the wall axes relative to a corner, but we could also define the same location using a different origin and measured along the “global” East-West, North-South axes.

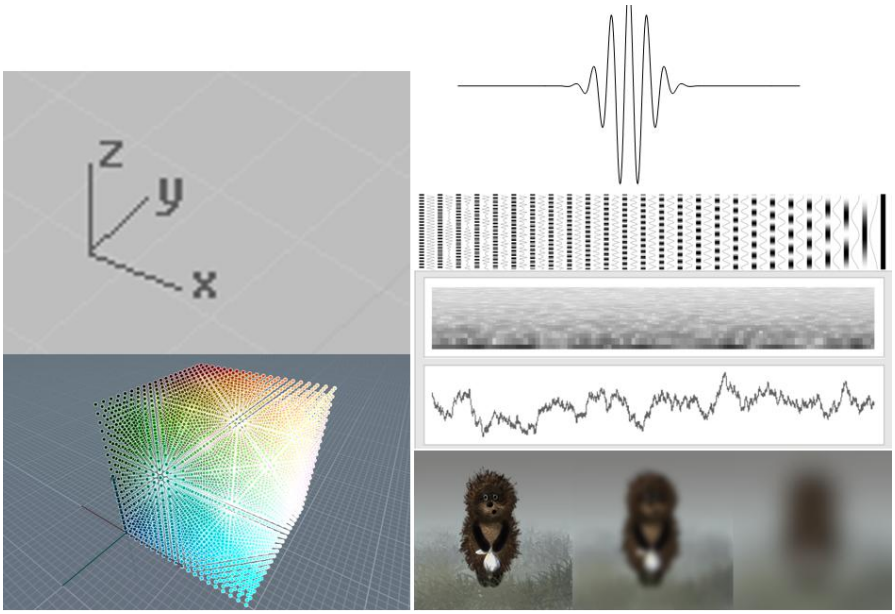


The pink point above has different coordinates depending on what coordinate system we measure it against. But we are talking about the same point. This will become important later as we can express the same exact object (image, text, etc) in relative to different systems.

Distance / proximity, metric

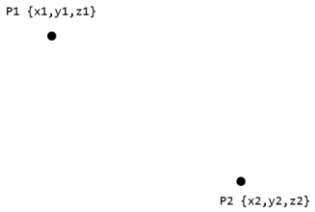
One of the most fundamental concepts in our experience of space is that of proximity which we quantify through the concept of distance.

The ability to measure distances between points is essential in defining what in mathematics is called a metric space. The term Geometry after all means “measuring the earth”.



The concept of distance between two points is also something that may depend on context. It does not have to be the standard Euclidian distance which measures the length of the straight line connecting two points. More exotic distance functions exist and in fact, any procedure that you make up that assigns a number to pairs of points (meeting some basic requirements, like the triangle inequality for example) can be considered a distance.

Below we review some alternative distance functions, which for three-dimensional space have limited practical use but they will become more important in the abstract spaces of machine learning.



Assuming we have two points p1 and p2 with coordinates {x1,y1,z1} and {x2, y2, z2} we can define the following distance functions:

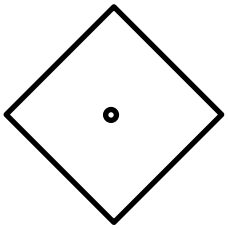
Manhattan distance

Manhattan distance is so called because it applies to spaces where diagonal movements are not possible, like the grid of Manhattan Island where one must move in a zig zag manner to reach a destination.

The Manhattan distance between two points p2 and p1 is given by the formula:

$$d = |x2 - x1| + |y2 - y1| + |z2 - z1|$$

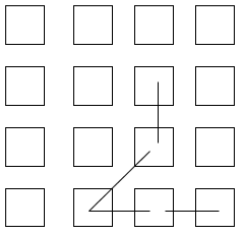
Once we have defined a distance function, we can also define geometric shapes. For example, the definition of a circle is not the intuitive “round shape” but the “set of points of equal distance to a centre point”. If we were to apply the latter definition to a space with the Manhattan distance a circle would look like the following figure:



In ML it is related to L1-norm and L1-Loss functions

Chebyshev distance

The Chebyshev distance is applied to spaces where roughly speaking moving diagonally from (0,0) to (1,1) would take the same time as moving from (0,0) to (1,0).



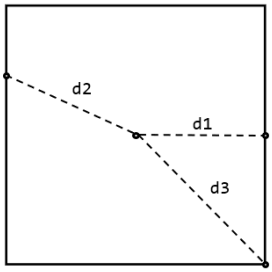
It is defined as follows:

$$d = Max(|x2 - x1|, |y2 - y1|, |z2 - z1|)$$

It says that when measuring the distance (difference) between two points, it equals the maximum separation along any of the three axes.

The Chebyshev circle looks like this:

d1 == d2 == d3

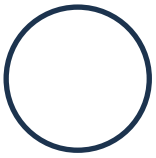


Euclidean Distance

The most common mathematical definition of distance when describing our three-dimensional space is the Euclidean distance which is given by the formula:

$$d = \sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2}$$

If we use this definition of distance, then a circle will look like our familiar round shape:



Keep in mind this formula, where we sum the squares of the differences of the coordinates. This is related to the Mean Square Error and L2-norms used in ML.

Minkowski distance

Minkowski distance is a generalization of the Euclidean distance. While in the Euclidean distance we sum the squares of differences of the point coordinates and then take the square root of these differences in the case of the Minkowski distance we raise each difference to a power p and then take the p_{th} square root of the sum of these differences.

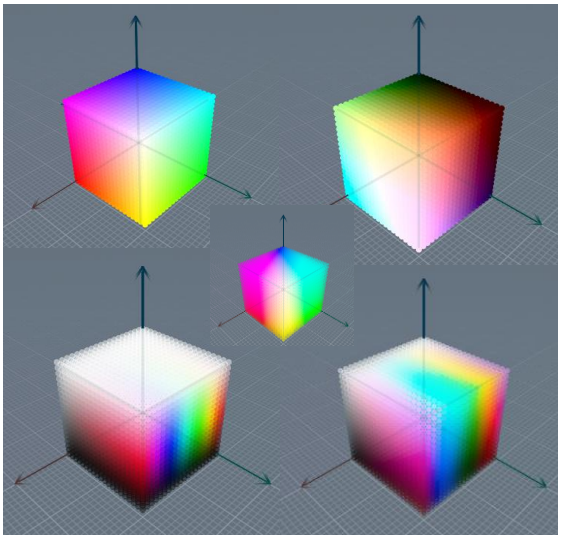
$$d = (|x2 - x1|^p + |y2 - y1|^p + |z2 - z1|^p)^{\frac{1}{p}}$$

```
Some special cases are:
for p=1 we get the Manhattan distance
for p=2 we get the standard Euclidean distance
for p=∞ we get the Chebyshev distance
```

It is related to the L-p norm in ML.

When using any of these distance functions in general the higher the power the more extreme differences are penalized. For example comparing the pixels of two images the L2 where we square differences per pixel will penalize (yield a bigger value) when we have a few pixels with very large differences than many pixels with smaller differences. This has subtle influence on what it means for two objects to be almost identical.

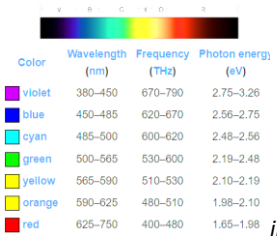
Colour Spaces



An interesting example of a space that has a few dimensions but is abstract is Colour space. In computer graphics and colour theory we can organize the totality of all colours that the average human can perceive or all the colours that a device can generate into a sort of space, where each colour is given a unique position and colours that are similar appear close by.

That means that we have a mechanism to assign a triplet of numbers {x,y,z} or {r,g,b} or {l,a,b} to each colour. Ideally, we want these numbers to yield some useful information about the colours they represent. For example, we would like it to be so that when we measure the distance between two colours we get a number that maps to our perceived notion of colour difference.

Colour is related to the wavelength of light.

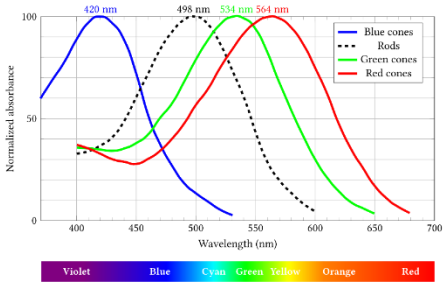


Color	Wavelength (nm)	Frequency (THz)	Photon energy (eV)
violet	380–450	670–790	2.75–3.26
blue	450–485	620–670	2.56–2.75
cyan	485–500	600–620	2.48–2.56
green	500–565	530–600	2.19–2.48
yellow	565–590	510–530	2.10–2.19
orange	590–625	480–510	1.98–2.10
red	625–750	400–480	1.65–1.98

img src: wikipedia

If humans could measure wavelength alone, we would need only one number (the wavelength) to describe colour.

But human vision does not measure individual wavelengths. We also perceive intensity (the number of photons) as brightness, and we average the effects over many photons to determine saturation.

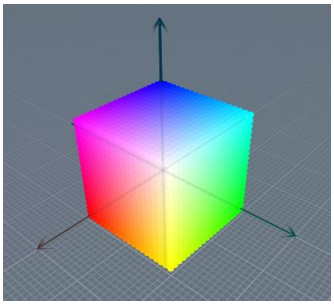


img src: Wikipedia

The human eye has three types of cone cells with photosensitive proteins that are sensitive to different bands of the electromagnetic spectrum. You see that their sensitivity is not evenly distributed but rather the green and ren sensitive pigments have a lot of overlap. So, our perception of colour distribution and intensity is not linear along the spectrum. That's why a yellow beam of light with the same number of photons will appear brighter than a blue one since the first one almost maximally activates two types of cells. Our trichromatic vision has other effects. For example, many mixtures of light wavelengths are perceived as the same colour. We cannot tell the difference between a monochromatic yellow laser and a mixture of red and green light although physically they are completely different. We also have concepts of perceptual colour that do not map to a single wavelength of light. Purple for example (is not violet) is a mixture of blue and red photons, there is no purple laser because there is no monochromatic single wavelength light that could have this effect (stimulate both blue and red cells simultaneously).

The other effect of our trichromatic vision is that colour is perceived through the mixture of three components and therefore we need at least three numbers to describe all possible perceived colours. That makes perceptual colour space inherently three-dimensional.

Most of you are familiar with RGB colour.

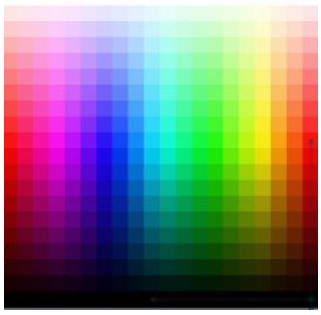


You can think of RGB space like a cube with Red, Green and Blue intensity axes. Black colour is at the origin of the coordinate system.

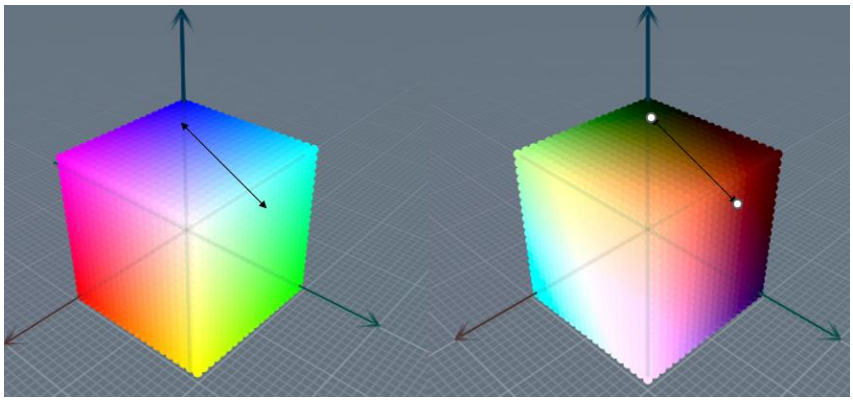
This system does not describe perceptual colour well. Its intended purpose is to describe generated colour. That is to control devices like computer monitors. The rgb values map to electronic current changes that control the intensities of small LEDs, or the masking opacity in of tiny filters.

If we increase a blue and a green LED's brightness by one unit each the human brain will perceive the green as having brightened significantly more than the blue one. So one step in the device colour space is not the same as one step in the human perceptual colour space.

You can see this clearly when you visualize a slice through the HSL (Hue Saturation Lightness) colour space which is a cylindrical version of the RGB space:

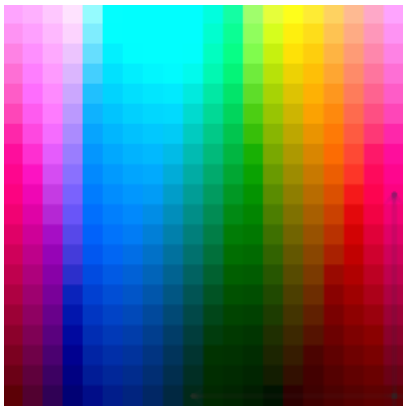


All the quads here represent the same step change in coordinates and all colours in a row should have the same brightness. However perceptually you can see some steps more clearly than others and the green yellow region appear significantly brighter.

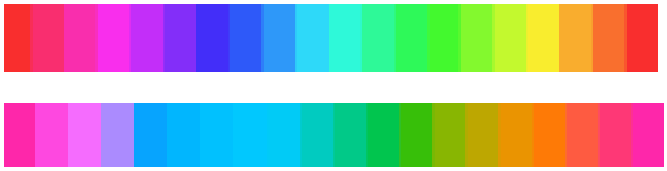


We want to be able to measure distances between colours and those distances to yield numbers that map to our perceived sense of similarity.

For accurate perceptual colour reproduction there are a few more accurate colour spaces like XYZ, Lab and their derivatives. One of those is the HCL (Hue, Chroma Luminance) which is another cylindrical space like HSL but is derived from the more perceptually accurate Lab space.



Above you see a slice at constant maximum chroma of the HCL model. Isolating a slice of HSL and HCL we can see the difference:

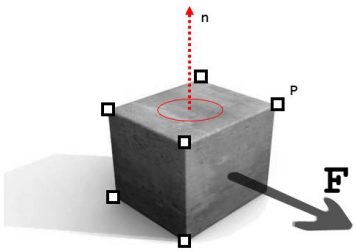


In the HCL version the blues do not stand out as particularly dark and the green yellows as particularly bright. Rather the perceived brightness is consistent.

The system is not perfect. There are some reasons for it, for one there are colours we can perceive (And these models capture) that most devices (computer monitors) cannot reproduce.

Vectors

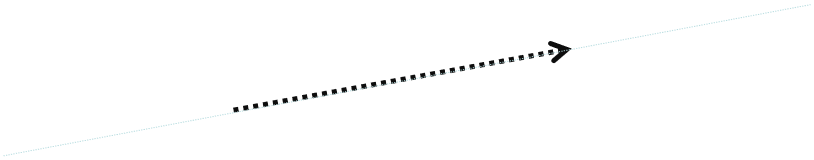
P(x,y,z) point
F(x,y,z) force
n(x,y,z) area element



The building blocks of geometry are points and vectors.

If locations are described by points, then the action of moving from one location to another is represented by another type of mathematical object called a “vector”.

You may have encounter vector quantities already in physics and math classes, the velocity of a car for example, the tangent of a curve.



a. Direction - the line of movement
b. Sense - forward or backwards along the line
c. Magnitude - Length, distance or force

A vector possesses three qualities, 1. a direction, 2. a sense and 3. a magnitude

We will often visualize vectors as arrows although this can be misleading in certain situations.

We will use vectors whenever we want to describe a movement or the tendency to move, or a direction in space.

Vectors as abstract displacements

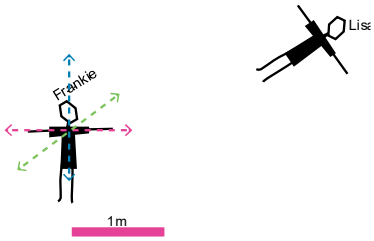
One of the difficulties in grasping the concept of a vector for students who first encounter it is that a vector can describe a movement in space in abstract without a start or end location defined. From now on we will use the term “translation” when referring to the movement of a geometric object from one location to another.

Linguistically there is an analogy we can make. If a point represents a location or an object, a vector corresponds to a verb that acts upon objects.

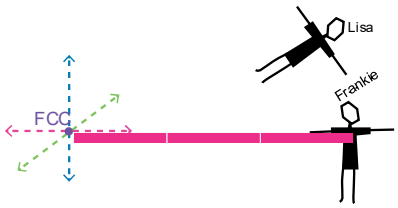
The instruction **"move left"** has no effect by itself unless it is directed to something that can move (an object, a person) and that has a notion of left and right. It also does not say by how much to move. Let's make this more precise:

"**Frankie**, move to **your** left **by** 3 meters".

That's a vector.



Frankie belongs to the group of animals "bilateria" so they have bilateral symmetry and hence define a local coordinate system with a left/right, front/back and up/down directions. Frankie also possesses a measuring convention (the meter) that enables them to compare lengths in space. We are also instructing them to move to their "left" so relative to their own coordinate system and not that of Lisa. Finally, Frankie is an object in space with a location that establishes an origin for their coordinate system and a start for their movement. To apply the translation "**Frankie**, move to **your** left by 3 meters" , Frankie needs to look to their left and place three meter sticks next to each other moving at the end of each stick. The word "by" is very important because it designates a movement relative to where they are now:

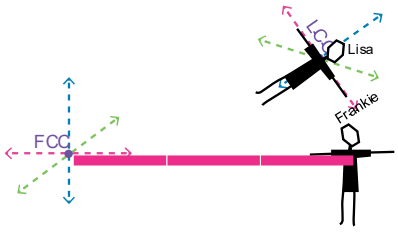


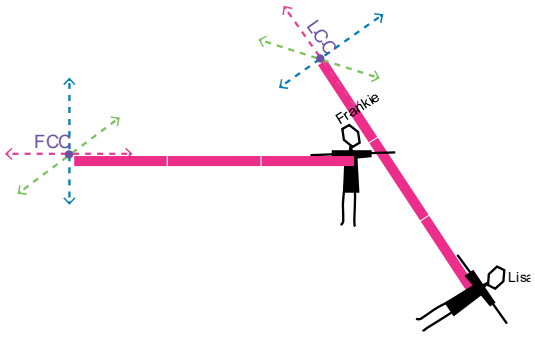
We will represent the instruction Frankie move to your left by 3 meters as the vector:

[**3m**, **0**, **0**]@FCC

This describes the translation along the Right/Left axis of the FCC (Frankie Coordinate System) by 3 meters. We can give the same instruction to Lisa.

"**Lisa**, move to **your** left **by** 3 meters"



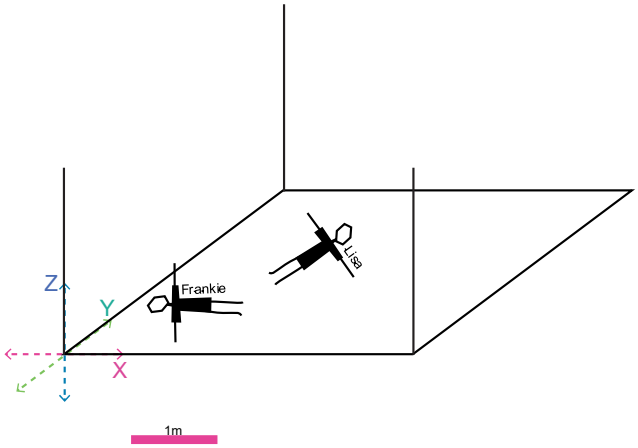


Lisa has their own coordinate system so the instruction is :

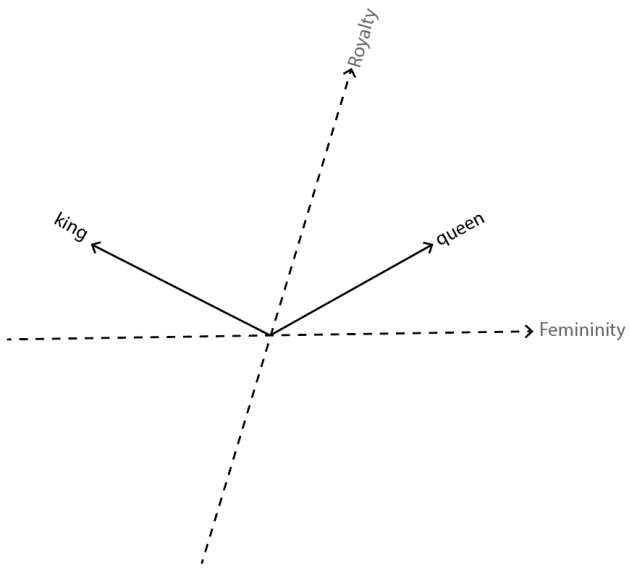
[3m, 0, 0]@LCC

When each object has its own “local” coordinate system we can still move them around in space and relative to each other. You will see this in systems that use a scene graph like game engines (unity3d, 3ds max etc…). But this coordinate system changes after each rotation and translation of an object because it is attached to the object that moves. That means that the concepts Left, Right, Front, Back, Up, Down are neither constant in time nor consistent between objects.

In CAD software we usually establish a global absolute coordinate system so that the instruction “move to the right by x units” has a global meaning and results in parallel translations for all objects it is applied to. It is like assuming a static room and measuring all translation along its edges. We usually assign the labels XYZ to the three directions and usually have some conventional way of determining whether positive x or negative x is left or right etc… But after we agree on that we can unambiguously start moving things around.



The axes X,Y and Z are called the basis vectors. There is nothing special about them they are just one of an infinite number of triples of directions in space we could have picked situationally. The establish the concepts of right-ness, front-ness and up-ness. The choice depends on what problem you try to solve. For example for a building project you may chose them so that they align with the global E-W, N-S direction or with some local condition (boundary of building site, major circulation axis, etc…).

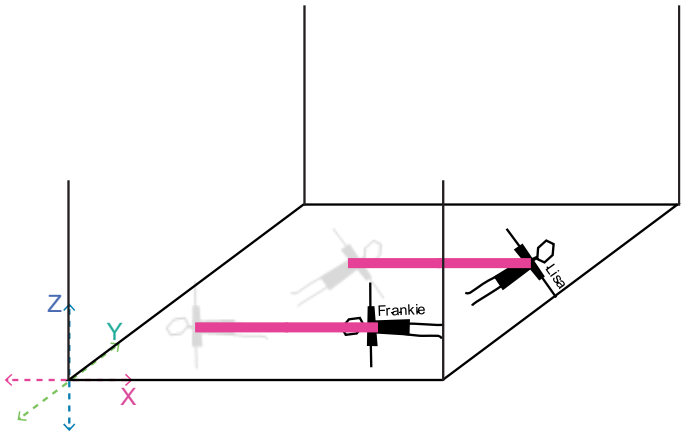


For a language model we can construct axes that may have specific semantic meaning.

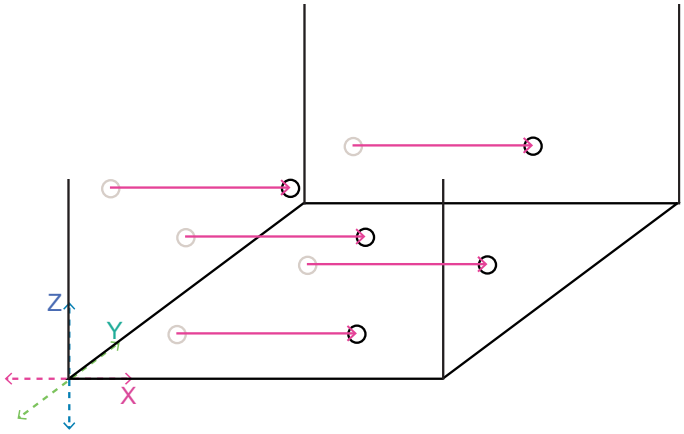
Vector operations

When a vector is used to represent a translation in such a way, it is not bound to any specific location in space. So, the vector

[3m, 0, 0] (that is the instruction move right by 3 meters) can be applied to millions of locations and they will all move parallel to each other by the same amount:



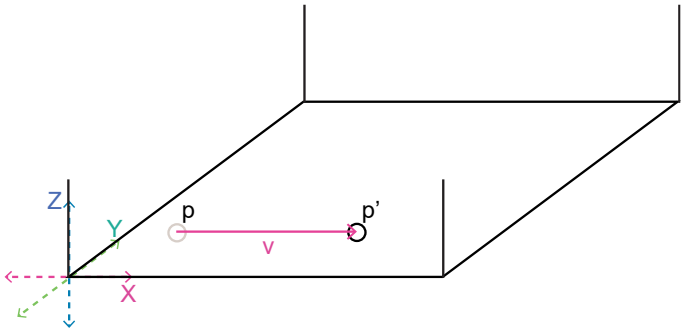
This is the important thing to take from here. A vector represents an action and has no point of application. The same vector can be "applied" to different geometric objects to translate the points that make them up:



All the arrows above represent the same vector applied to different objects.

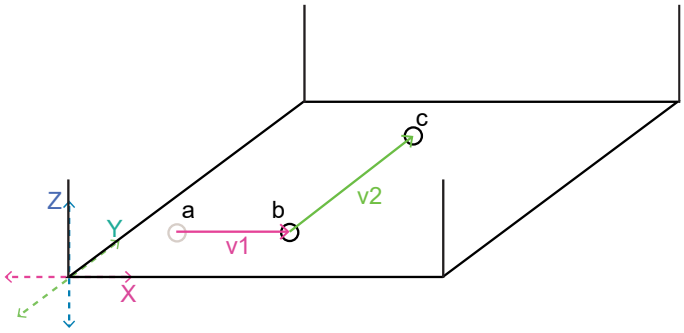
We will also use the + operator to represent the application of a vector as a translation v to a point p to yield a new point p':

```
p' = p + v;
```



Now what if we were to give a composite instruction like:

“move **a** by 2m along X to get **b**” **and then**
“move **b** by 3m along Y to get **c**”

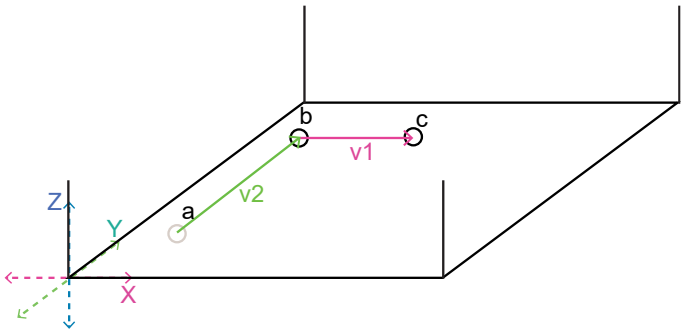



```
b=a+v1
c=b+v2
```

with v1:[**2**, **0**, **0**] and v2:[**0**, **3**, **0**]

What if we changed the order to:

“move **a** by 3m along Y to get **b**” **and then**
“move **b** by 2m along X to get **c**”

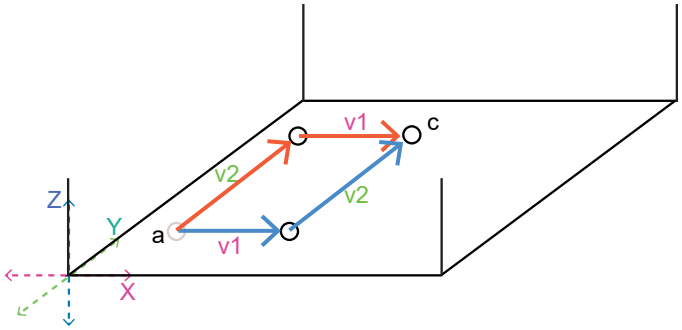



```
b=a+v2
c=b+v1
```

It seems that the order of movements does not matter because we always land in the same spot. Then we could write:

```
c = (a + v1) + v2
or
c = (a + v2) + v1
```

and we would get the exact same result c.

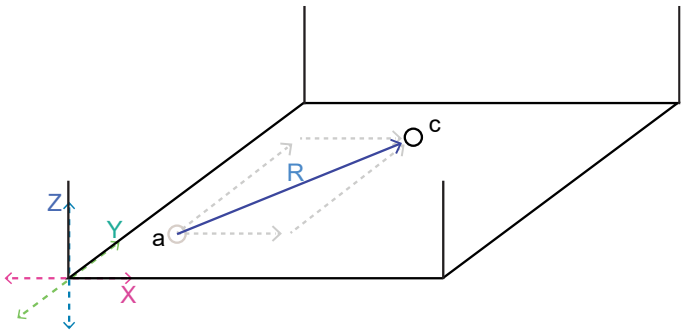


This suggests that the blue path (v1+v2) is equivalent to the orange path (v2+v1).

with v1:[2, 0, 0] and v2:[0, 3, 0] we can give a shorter instruction:

```
c = a + (v1 + v2)
```

“move **a** by 3m along Y and 2m along X to get **c**”



We will call this the resultant vector R with:

```
R=v1+v2
```

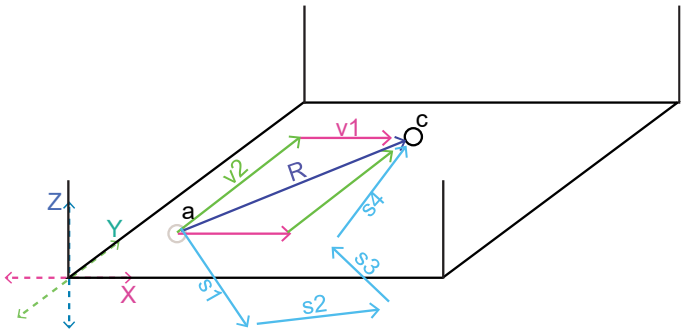
And v1:[2, 3, 0]

> *This simple formula above contains a very powerful idea. That we can treat actions as objects and do math with them without having to apply them to concrete geometric objects like points first. We also see geometrically that combinations of translations and hence the sums of the vectors that represent them are commutative A+B=B+A and the order of operations does not affect the result.*

v1 and v2 represent two actions of translation and they are added together to yield another “longer” action of translation. Then we can apply the result R to any number of objects to move them along R.

Vector components

We use numbers to measure lengths. By measuring lengths along some axes and from a point (a coordinate system and its origin) we can identify locations in space as points, the distance along each one of the X, Y and Z axes representing one coordinate of the point. We can also use numbers to measure displacements in space again along the X, Y and Z axes, and these are the components of vectors. Point coordinates are always defined in relation to some origin point (10 meters along x **from 0**) while vector components are pure differences (10 meters along X). In mathematics points identify locations in a space while vectors represent displacements or differences.



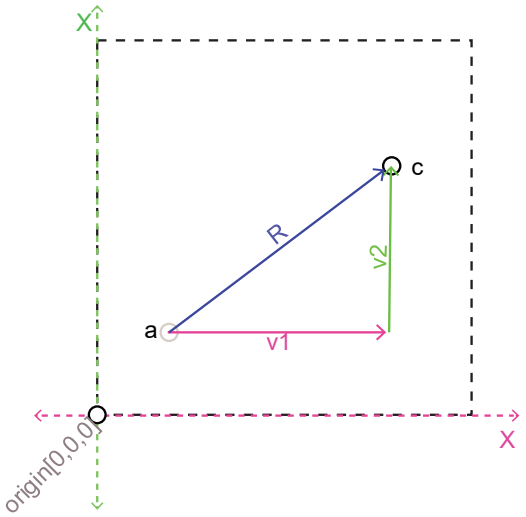
We can compose any number of translations together. In the above diagram:

R = (s1+s2+s3+s4) = (v1 + v2)= (v2 + v1)....

It simply states that to go from **a** to **c** we can follow any of the above paths:
R, (s1+s2+S3+S4) , (v1+v2) or (v2+v1)

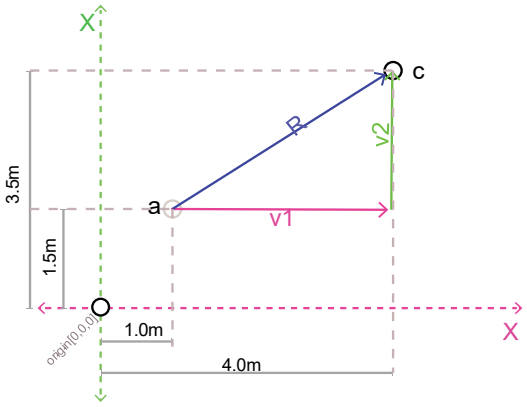
R is of course the simplest "direct path" so it expresses the translation from a to C more concisely. Given two points a and c there is a unique single vector R that can move from a to c.

Let's move into 2d for a moment and look at the diagram from above:

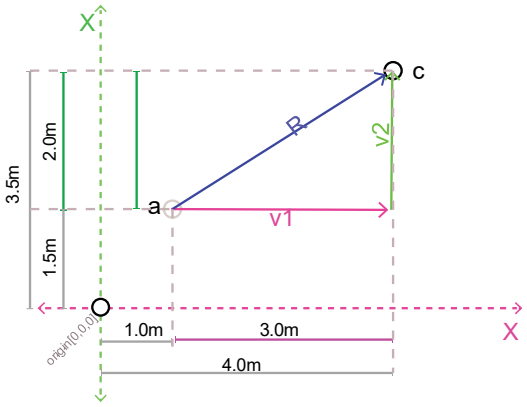


Given a vector R there is a unique set of vectors (2 in 2d and 3 in 3d) that are parallel to the X, Y and Z axes and when added together they yield R. So, from all the vectors that we could add together to get R, v1 and v2 are special because they are parallel to the coordinate axes that we established.

The lengths of these two vectors are called the components of R in the current coordinate system.



Given the point a with coordinates [1.0, 1.5, 0.0] and the point c with coordinates [4.0, 3.5, 0.0] R is the vector that represents the direction and magnitude of movement to reach c from a. To assign to this movement (vector) some numerical values we use its decomposition to axis aligned vectors (v1+v2). The amount of displacement along X axis (vector v1) will be the **X component** of the vector R and the amount of displacement along the Y axis will be the **Y component** of vector R.



So the vector R represents the combined translation of 3m along the X axis and 2m along the Y axis and we will write it in component form as:

R=[3, 2, 0]

Notice that points have "coordinates" vectors have "components". Coordinates represent addresses in space measured along some curves (that may or may not be straight lines) from a reference point called the origin. The components of a vector R are partial translations along the axes of the coordinate system that together would recreate the vector R but they don't "exist" at a particular point in space. They are pure differences.

In the above example the vectors have the following components:
R[3,2,0]
v1[3,0,0]
v2[0,2,0]

and the points have the following coordinates:

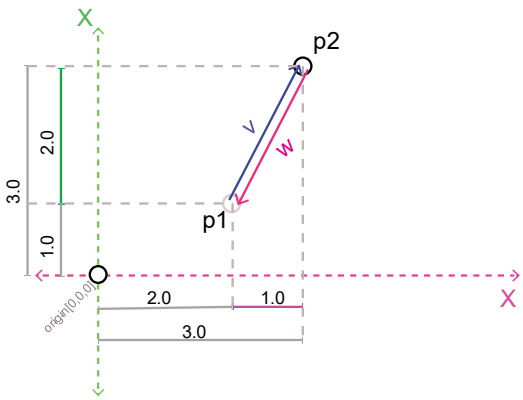
a[1, 1.5, 0]
c[4, 3.5, 0]

The Directions X axis and Y axis represented by unit length vectors [1.0, 0.0, 0.0] and [0.0, 1.0, 0.0] are special because we use them in the first place to establish our coordinate system and are called "basis vectors". They enable us to express any direction in space as a composite of these special directions. These two directions don't have to be orthogonal to each other or even have the same units but in most cases they will be.

Below we are going to review the fundamental operations between vectors and points that will enable us to build most advanced computational geometric operations. You can seek more proofs and mathematical treatment in any textbook on analytical geometry but for now we are going to cover everything with no prior knowledge required and just enough to get us going with programming.

Reverse Vector: -vector

Let's look at two points p1 and p2 and the vectors that express the translations between them. Geometrically going from p1 to p2 is not the same as going from p2 to p1 and the mathematics reflect that:



v = p2 – p1 = [3,3,0] – [2,1,0] = [3-2, 3-1, 0-0] = **[1, 2, 0]**
w = p1 – p2 = [2,1,0] – [3,3,0] = [2-3, 1-3, 0-0] = **[-1, -2, 0]**

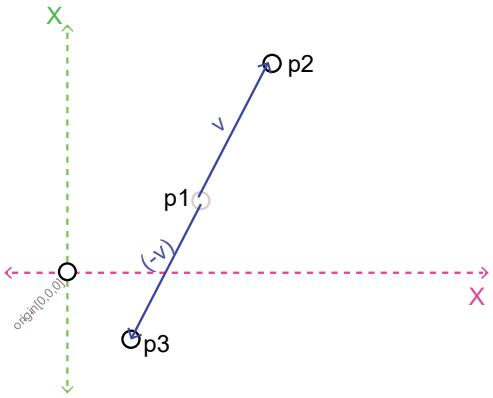
Although the two vectors have same length and lie along the same line, they have opposite sense (they point in opposite directions). Numerically it means that if a vector has components [dx, dy, dz] then its reverse will have the components, [-dx, -dy, -dz]. Also:

v + **w** = **[1, 2, 0]**+ **[-1, -2, 0]** = [0, 0, 0]

that is if we start at any point and move first along v and then along w we will return to the same point (0 translation).

Since vectors possess a concept of the reverse, we can ask what happens if we add the reverse of a vector to a point that is, if we "subtract" a vector from a point:

p3 = p1 **– v** = [2,1,0] – **[1, 2, 0]** = [1, -1, 0]

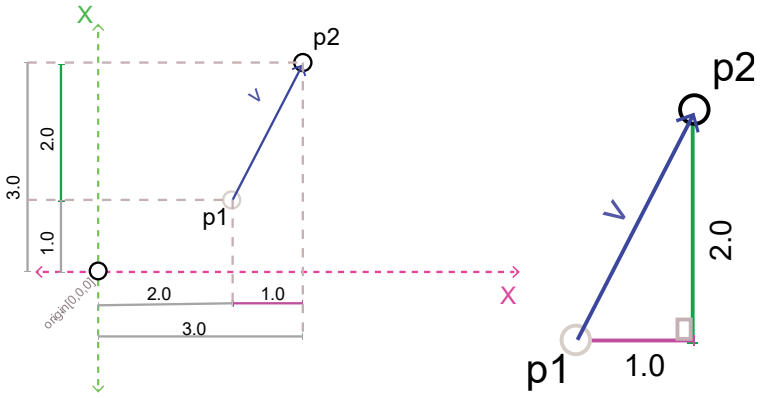


That is, we moved point p1 to the opposite direction of v to reach point p3.

magnitude, length, norm of a vector : |v|

Given a vector we might want to know how much it would displace a point without caring about the direction of movement. That is, we are asking for the distance by which a point would travel if we were to add the vector v to it. We will use three terms for this distance almost interchangeably: **length**, **magnitude** and **norm**. The choice depends on the context. For example, for vectors representing spatial translations we will mostly use the term length, for vectors representing velocities or other differential properties we will use the term magnitude and for more abstract situations we will use the term norm. The norm of a displacement vector is its length, or the norm of a velocity vector is the speed.

Going back to our example of moving from point p1 to point p2:



The length of vector v can be given by Pythagorean theorem as :

$$|v| = \sqrt{dx^2 + dy^2}$$

We will use the symbol |v| to stand for the norm or length of a vector.

This formula extends to any number of dimensions. So, for three-dimensional space we would have:

$$|v| = \sqrt{dx^2 + dy^2 + dz^2}$$

In the above example the length of the vector v [1,2,0] would be:

$$|v| = \sqrt{1^2 + 2^2 + 0^2} = \sqrt{1 + 4} = \sqrt{5} = 2.236067977499789696409\ldots$$

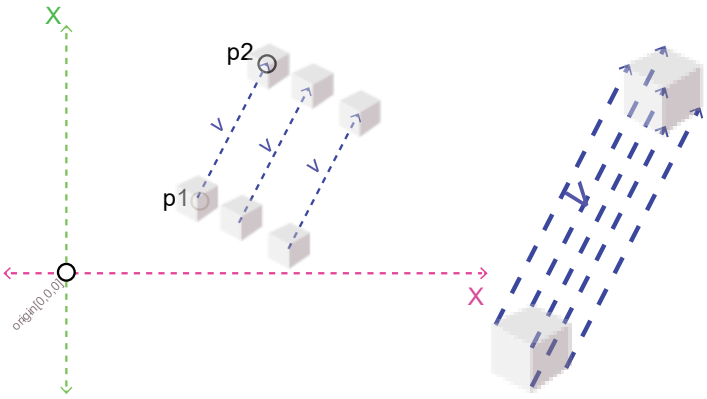
The length of a vector is of course always a positive real number and in some sense is a generalization of the concept of the absolute value of a number in more than one dimensions, hence the |v| symbol.

Scaling vectors : (vector * number)

In the previous example we saw how to calculate the vector that describes the translation required to move an object from the location of a point p1 to the location of another point p2.

v = p2 - p1

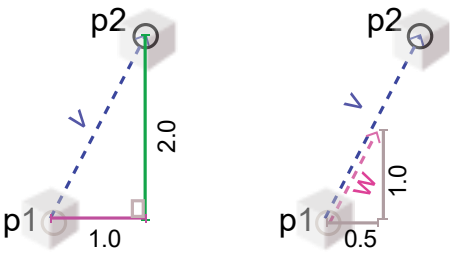
If we had two boxes and we were to move them from p1 to p2 we could apply the vector v on them (basically add the vector to each point of each box) and they would move along v:



What if we wanted to move the boxes half-way between p1 and p2. That is move along the vector v but only by half its length.

For that we would need to construct a new vector w which has the same direction as v but only half its magnitude.

In vector arithmetic this is done by multiplying a vector with the desired scaling factor. Let's look at a concrete example:



If we multiply the vector v by the number 0.5 then we get a new vector w:

w = **v***0.5 = [1,2,0] * 0.5 = [1* 0.5, 2* 0.5, 0*0.5] = [0.5, 1, 0]

notice that multiplication of a vector v[dx,dy,dz] by a number s is distributive over the components :

v *s = [dx*s, dy*s, dz*s]

Which simply means we multiply each component of the vector by the same number. The result of this operation is a new vector that has the same direction as the original, but its length is “s” times the length of the original vector v.

So for our numeric example we saw that the length of v was:

|**v**|=2.24…

So the length of w is :

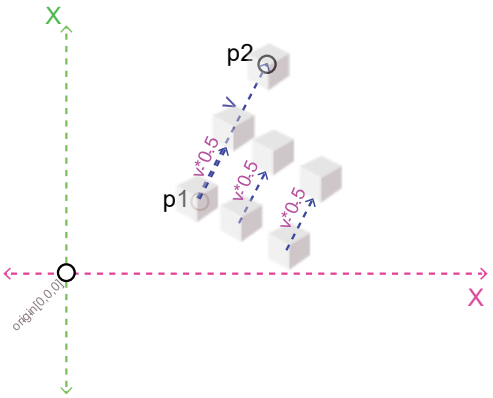
|**w**| = |**v***0.5| = |**v**|*0.5 = 1.12…

You can do the math given a vector v[dx,dy,dz] and a number s :

$$|w| = |v*s| = \sqrt{(s*dx)^2 + (s*dy)^2 + (s*dz)^2} =$$

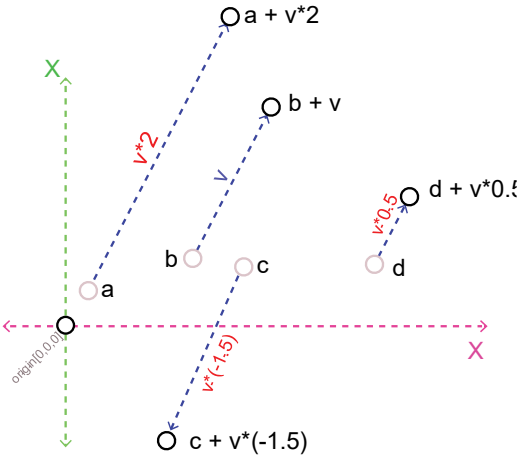
$$\sqrt{s^2*(dx^2+dy^2+dz^2)} = s\sqrt{(dx^2+dy^2+dz^2)} = s*|v|$$

So, in our initial example if we add v*0.5 to all the boxes we will move them halfway along v:



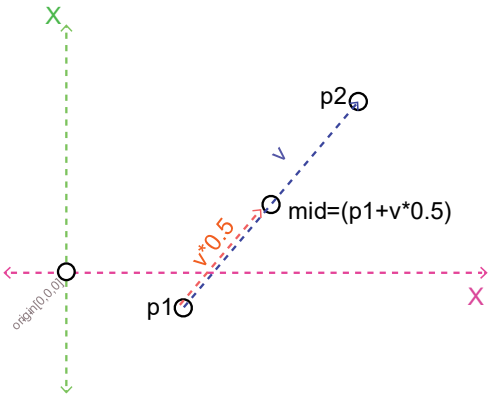
In general, geometrically speaking translations are additive while scaling is multiplicative in nature.

We can also scale a vector by negative values or values larger than 1.0. Here is an example of adding the same vector v multiplied by different factors to 4 different points a,b,c and d:

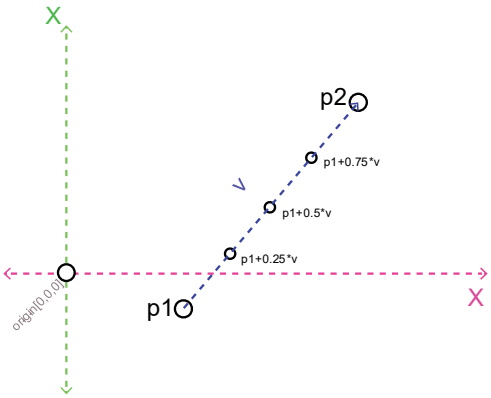


In that sense we can see that when we reversed a vector to get its opposite pointing one, it would have been the same as multiplying the vector by (-1). {there is a typo in the diagram for the negative vector it should read v*(-1.0)}

Given two points p1 and p2 and the vector v=p2-p1 we can generate any point in between them by taking advantage of this. The midpoint for example is the point we get if we travel halfway along v from p1:



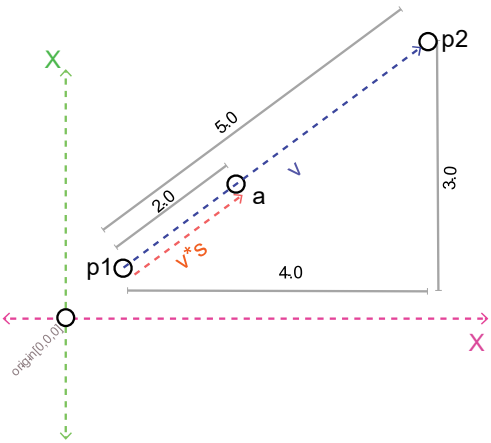
And this generalizes to:



We will see later in the semester how this “linear interpolation” between points establishes the parametric equation for a line segment.

normalization, unit vectors : $vector * (\frac{1}{|vector|})$

In the previous example we saw how we can move along a vector using fractions of its length. This allows us to calculate any point along the vector from a starting point if we know the fractional length we want to move by. But what if we wanted to move by an absolute amount, say 2 meters from point p1 to point p2. We can of course do that by dividing 2 by the distance between p1 and p2 (the length of v) to get the fractional displacement. That is to place a point “a” at the 2m mark between p1 and p2:



The problem is we need to figure out what "s" should be so that v*s has a length of 2.

We know how to calculate the length of v, |v| (in the above example |v| is 5m).

So the fractional displacement s along v is:

$$s = \frac{2}{|v|} = \frac{2}{5} = 0.2$$

And so the point a will be:

a = p1 + v*0.2

But what if we wanted to place points at 1m 2m 4m and so on. There is a more efficient way to express something like that by noticing that each time we need to divide the displacement we want by the length of the vector itself |v| to get the fractional displacement. Wouldn't it be better if we had a vector "n" that comes pre-divided so we could simply write:

a = p2+n*2

Without having to divide by the vector length each time. Such a vector does exist and is simply the vector along v (same direction and sense) but with a length of 1.

In many applications we will need to construct a special vector for which we are interested only in its direction but not its length. We usually use a process called normalization of the vector where we calculate a new vector with the same direction but a length of 1.0. Vectors with length 1.0 are called "**unit vectors**" and the process of creating them from other vectors is called "**normalization**" or occasionally "**unitization**".

We saw before that if we have a vector v we can change its length without affecting its direction by multiplying it by a number s:

n = **v***s

we also know that when we do that we get :

|**n**| = |**v**|*s

That is the length of the new vector n is equal to the length of the original vector v multiplied by the factor s.

So what is the number s that if we were to multiply it by |**v**| it would yield 1.0? That's of course the inverse of |**v**|:

For |**n**| to be 1.0 "s" has to be (1/|**v**|).

So normalization is the procedure where we multiply a vector by the inverse of its length to produce a new vector:

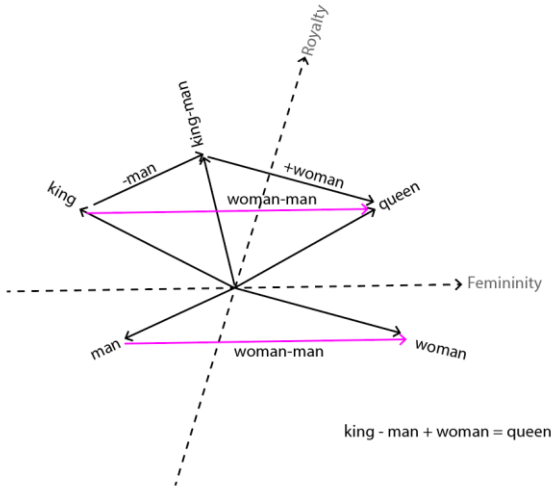
n = v * (1/|**v**|)

In Machine learning we will encounter vectors whose magnitude does not carry any useful information and very often we will just normalize them. For example in some language embeddings each word is represented by a vector and the magnitude of a vector is proportional to how often the word appeared in text. When comparing

words for semantic similarity this information is often not important as there are similar words (synonyms) where one is much more common than the other.

In addition, normalization of abstract vectors may be related to the removal of other information that would make comparisons difficult. If a vector represents a piece of audio, then normalization may make the volume consistent when comparing snippets of sound recorded under different conditions.

Vector operations in semantic space



The above discussion regarding what it means geometric to add and reverse vectors extends to more abstract spaces, like the ones defined by language embedding models. In such models each word is represented by a vector. Words that have similar semantics point in similar directions. This space does not have predefined axes with fixed meaning. Instead, if we wanted to find an axis we need to construct it from known words.

One common test for word embedding models is to see whether they capture analogies. E.g. find the word that relates to woman in the same way that king relates to man. Another way to state this is take the word king and move it along the man->woman axis and see where it lands. Many models will yield the term queen.

If you think of Man, Woman, King and Queen as vectors. Then the Woman-Man creates a new vector that roughly speaking points in the direction of increasing femininity. If we add this vector to any other concept we are translating towards that direction. That's the sense in which the equation:

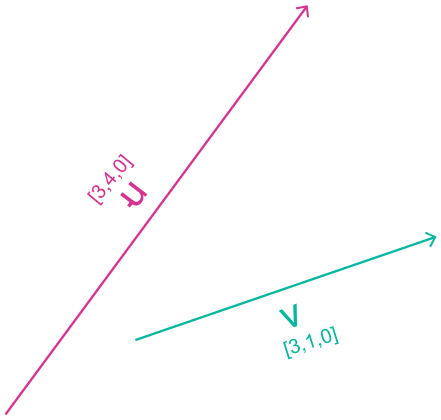
King – Man + Woman -> Queen

Or reformulated as:

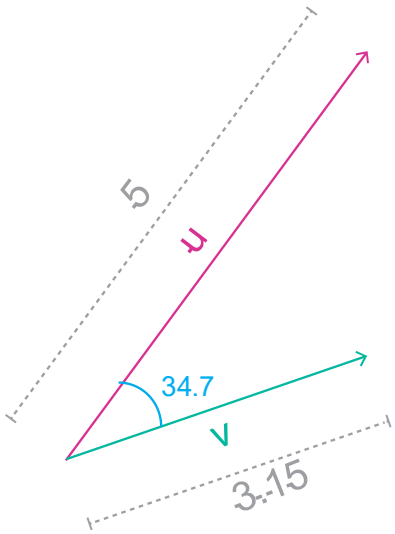
King + (Woman-Man) -> Queen

The dot product

Let's start with two vectors u and v :

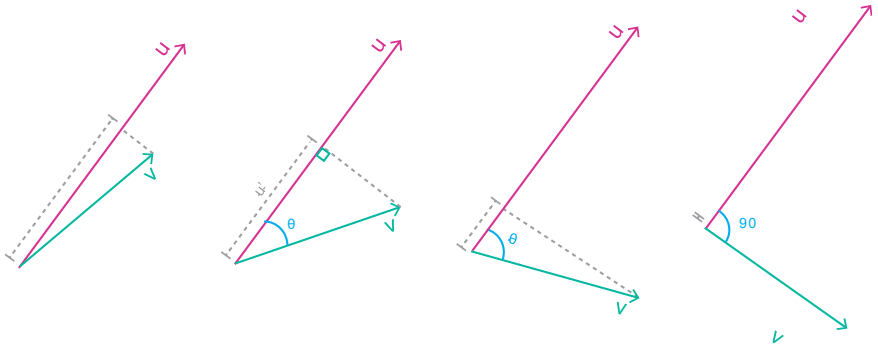


How do we compare vectors? For two numbers a and b we can form the difference |a-b| and that will tell us how similar they are. For two vectors the situation is more complex because we would have to compare the lengths and the angles of the vectors.



Just comparing the lengths will tell us something about the magnitude difference but not the difference in direction that in certain contexts might be more important. It is as if the number 5 and number 3.15 exist along different lines in space (the occupy different subspaces).

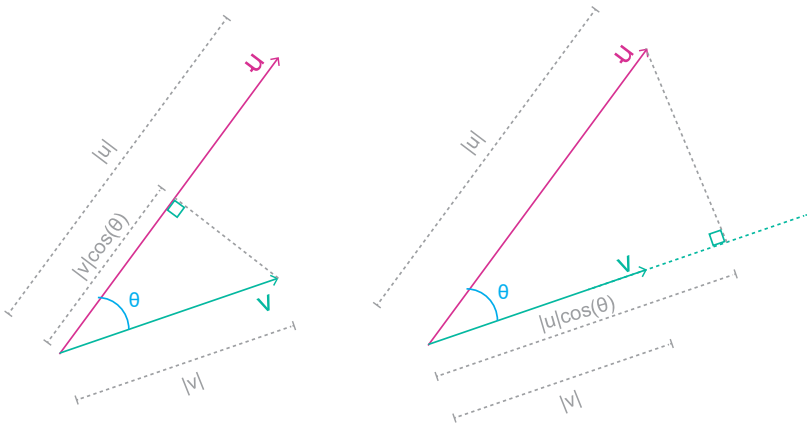
One way to compare things is to project them onto the same space. Let's project v on u (we will also use the norms |u| and |v| for the lengths of u and v from now on and θ for the angle between them):



We can see here that the projection of v on u changes in length with the angle. So, it is proportional to the length of v but also inversely proportional to the angle. It becomes 0 when u and v are perpendicular to each other and is maximal when they are parallel. So somehow for two vectors, pointing at 90 degrees, is as different as they can get regardless of their lengths.

What the projection measures is how much of u-ness (the quality of u) exists in the vector v. You can think of difference in direction as qualitative and difference in magnitude as quantitative.

Let's see what that projection is like numerically:



By the definition of the cosine:

$$\cos(\theta) = \frac{adjucentSide}{hypotenuse} = \frac{uProjection}{|v|}$$

and so, the projection of v on u is |v|*cos(θ) and of u on v is |u|cos(θ)

We see that these two expressions change with the angle between the vectors because of the cosine but they are not symmetric that is v on u is not the same as u on v. We can multiply the projection of v on u with the length of |u| or reciprocally the projection of u on v with the length of |v| itself to get a more symmetric expression:

$$|u|*|v|*\cos(\theta)$$

This is symmetric, and is neither the projection of u on v or of v on u. It gives as a single number that combines both the length and angle information encoded in the vectors. This number will be very large if both vectors are large and point in the same direction. If one of the vectors is a unit vector (ie |u|=1) then this expression is indeed the projection of the other vector on the direction of the unit vector. If both vectors are unit vectors (i.e. |u|=1 and |v|=1) then this expression depends only on the angle between them.

We will call this expression the dot product between the vectors u and v and use the

"." Dot symbol as the dot product operator:

$$\mathbf{u.v = |u|*|v|*\cos(\theta)}$$

the dot product of two vectors is a number with the following geometric properties:

a. It is 0 if the two vectors are orthogonal

b. It becomes maximum when the two vectors point in the same direction

c. It becomes minimum (negative) when the two vectors point in opposite directions

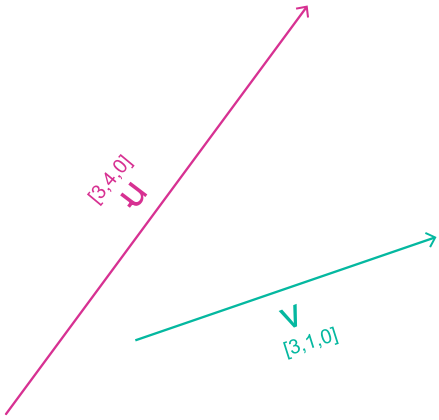
d. It is proportional to the lengths of both vectors

Of course, this creates a problem of how to quickly measure the angle of two arbitrary vectors in space. Luckily because we are working with cartesian coordinates there is a very simple expression of the dot product. For two vectors u[ux,uy,uz] and v[vx,vy,vz] the dot product can be calculated as:

$$u.v = ux*vx + uy*vy + uz*vz$$

This is an easier and faster expression to compute since it only involves the XYZ components of the vectors and is numerically equal to $|u||v|\cos(\theta)$.

Going back to our original example:



$$u.v = ux*vx + uy*vy + uz*vz = 3*3 + 4*1 + 0*0 = 13$$

We can also use it to compute lengths of vectors. If we take the dot product of a vector with itself then we simply get the square of its norm:

$$u.u = ux*ux + uy*uy + uz*uz = ux^2 + uy^2 + uz^2 = |u|^2$$

And the square root of the dot product u.u is simply the length of u itself.

The dot product has the following algebraic properties (below we will use bold for vectors and regular font for numbers):

u.**v** = **v**.**u**

w.(**u** + **v**) = **w**.**u** + **w**.**v**

(s*__u__).(t*__v__) = s*t*(__u__.__v__)

If **u**.**v**==0 then **u** and **v** are **orthogonal**

Finally, we can use the dot product to quickly compute angles between any two vectors in space using only their components:

Since we know that:

u.**v** = |**u**|*|**v**|*cos(θ)

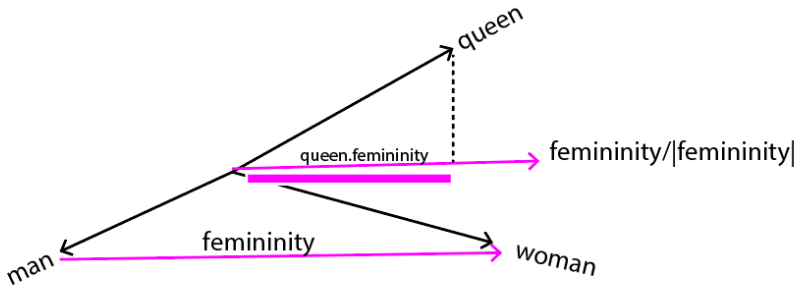
u.**v** = ux*vx + uy*vy + uz*vz

we can compute the cosine of the angle (and hence the angle) by simply inverting the first equation:

$$\cos(\theta) = \frac{\mathbf{u}.\mathbf{v}}{|\mathbf{u}|*|\mathbf{v}|}$$

We can easily compute u.v using the component form (ux*vx + uy*vy + uz*vz) and we know how to compute the lengths of vectors |u| and |v| using the Pythagorean formula.

The above formula extends to abstract vectors and in Machine Learning is called the cosine similarity measure.



To revisit our language embedding example. Let's say we wanted to measure how much femininity exists within the concept of queen. This means that we need to project queen onto the direction of femininity.

```
Fem = woman – man          (find the femininity direction from two known vectors)
F' = Fem / |Fem|           (normalize it to become a pure direction)
QF = queen . F'            (take the dot product to project queen on F)
```