Semantic Vectors

Joseph Juma
Independent
joseph@josephjuma.com

April 2026

ABSTRACT

An explanation of Semantic Vectors.

1 Semantic Vectors

When working with space or geometry it's common to use two-dimensional and three-dimensional vectors. By far the most common type of vectors are **Euclidean** vectors whose elements are real-numbers (i.e. vectors defined over $\mathbb{R}^2$ and $\mathbb{R}^3$). A Euclidean vector is one defined in a space where distance between two points is measured by the **Euclidean Distance Metric** which has the following formulas in (respectively) two-dimensional, three-dimensional and n-dimensional space:

$$\begin{aligned} \Delta(A,B) &= \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2} \\ \Delta(A,B) &= \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2 + (z_B - z_A)^2} \\ \Delta(A,B) &= \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \end{aligned} \tag{1}$$

Where...

1. $\Delta(A,B)$ is the distance between two given points A and B.
2. A and B are points in either two-dimensional, three-dimensional or n-dimensional real-number space.
3. x_A, y_A and z_A are respectively the first, second and third element of A.
4. x_B, y_B and z_B are respectively the first, second and third element of B.
5. b_i is the i-th element of B.
6. a_i is the i-th element of A.

In Python-style pseudocode this would look like:

```
def euclidean_distance_2D(A,B):
    """
        Given two, two-dimensional vectors, returns the Euclidean
        distance between them.
```

```
    """
    return math.sqrt((B[0] - A[0])**2 + (B[1] - A[1])**2)

def euclidean_distance_3D(A,B):
    """
        Given two, three-dimensional vectors, returns the Euclidean
        distance between them.
    """
    return math.sqrt((B[0] - A[0])**2 + (B[1] - A[1])**2 + (B[2] - A[2])**2)

def euclidean_distance_n_dimensional(A,B):
    """
        Given two, n-dimensional vectors, returns the Euclidean
        distance between them.
    """

    s = 0
    for i in range(len(A)):
        s += (B[i] - A[i])**2
    return math.sqrt(s)
```

These work well with software methods that can work with real-numbers, or those that work over **continuous domains** (i.e. sets of numbers that can be incremented by tiny amounts, and which have many numbers in them).

But not all systems work well with such domains. You might have methods that work better on **discrete domains**, which are those with distinct numbers that are incremented in large blocks (e.g. whole-numbers or integers). Additionally, you might have a system that works best with a small set of possible values that each have a strong conceptual meaning.

Because the word *semantic* refers to "meaning", we might call a vector defined over a small set of meaningful ideas a **Semantic Vector**. These are vectors which encode meaningful human ideas about space, rather than strictly mathematical ones.

A given semantic vector is comprised of two elements: a **semantic direction** and **semantic distance**. Euclidean vectors also have these ideas, but the direction is calculated by using the normalization formula on the vector (i.e. $||V|| = \frac{v_i}{\sum_{j=1}^{\#V} |v_j|}$) and the distance the previously mentioned Euclidean distance metric. Instead, in semantic vectors these are the actual values comprising the vector itself. So we might write a semantic vector with the following mathematical syntax:

$$A = (N, \Delta) \tag{2}$$

Where...

1. A is the semantic vector.
2. N is the semantic direction of the vector.
3. Δ is the semantic distance of the vector.

In a JSON-style pseudocode this would be:

```
semanticVector = { direction, distance };
```

With this established it's worth noting two things. First, the semantic vectors described herein are *technically* Semantic Euclidean Vectors, because they're going to assign semantic values that correlate to notions of distance and direction from Euclidean space. Second, while Euclidean vectors grow in size depending on how many dimensions they cover, in Semantic vectors the semantic direction is what grows in size instead.

To better understand these vectors, it will now be useful to cover both semantic direction and semantic distance.

1.1 Semantic Direction

A **Semantic Direction** is a value that describes some notion of direction in simple, meaningful terms. For example, in two-dimensions the semantic direction is usually just the cardinal (compass) directions: North, South, East and West, and their combinations: North-West (NW), South-West (SW), North-East (NE) and South-East (SE). In three-dimensions we add the vertical axis of Up and Down, which results in a total of 26 possible directions, which I won't iterate on herein but look something like: Up, North, North-Up, South-West-Down, etc.

It's also useful to have a "null-direction" to express when there is no direction at all, so we have in-total 27 semantic directions in 3D, and 9 in 2D. If we wanted to generalize this to n-dimensions it would be that we have 3^n directions.

However, because semantic directions are foremost useful because they are few in number, and encode things with meaning to human brains, it's rare to every go past four-dimensions (i.e. North, South, East, West, Up, Down, Past and Future). For the purposes herein, two and three dimensional will be those discussed.

1.1.1 Relationship to Euclidean Space

Semantic and Euclidean directions can be converted into each other, by assigning a given Euclidean axis to a pair of semantics. For example, "Up" and "Down" would be the z-axis in a Euclidean vector (i.e. the third value) with "Up" being the positive z-axis (i.e. $(0, 0, 1)$) and "Down" being the negative z-axis (i.e. $(0, 0, -1)$). This means that for each element in a Euclidean vector, there are two corresponding directions in a semantic direction.

So the semantic direction "North-West-Up" would convert to the Euclidean vector $(1, -1, 1)$, where the first value is "North/South" (North; 1), the second "East/West" (West; -1) and the third "Up/Down" (Up; 1).

A Euclidean vector can be converted to a semantic direction by converting the vector into a vector of all 1 or 0 values, then assigning it to the corresponding semantic direction. This conversion can be done with the following formula:

$$V' = \frac{v_i}{|v_i|} \tag{3}$$

Where...

1. V' is the Euclidean direction vector.
2. V is the Euclidean vector.
3. v_i is the i-th element in V.

In Python-style pseudocode this would look like:

```
def derive_euclidean_basis_vector(vec):
```

```
"""
    Given a Euclidean vector, returns a corresponding basis vector.
"""
basis = vec
for i in range(len(basis)):
    basis[i] = basis[i] / abs(basis[i])
return basis
# Compact alternative: return [(x/abs(x)) for x in vec]
```

For example, the 3D Euclidean vector $(5, -3, -7)$ would become the Euclidean direction vector $(1, -1, -1)$ which maps to the "North", "West" and "Down" semantics, so it would be the North-West-Down ("NWD") in semantic direction terms. The 2D Euclidean vetor $(-13, 0)$ would become $(-1, 0)$ which maps to "South" and "None" - so it's the "South" direction.

1.1.2 Software Design Implications of Semantic Directions

Unlike Euclidean vectors which grow by one real-number (i.e. 32-bit or 64-bit floating-point value) per-axis, semantic vectors increment by three distinct values per-axis. This allows a given semantic vector to only take up $\lceil \frac{log(3^n)}{log(2)} \rceil$ bits, which is far more efficient than a Euclidean direction vector.

The primary trade-off is in conversion to and from Euclidean vectors.

1.2 Semantic Distance

Semantic Distance is the use of numbers or meaningful words to convey notions of distance that have meaning to humans. Think of it like this: a human mind doesn't necessarily know the difference between 11 or 12 centimeters without numerical values, but does know the difference between something that's "close" or "distant". These notions of distance are *distance semantics*, and they're what's encoded within a semantic distance.

Generally, distances with regards to human semantics operate on "scales". Something like meters, tens-of-meters, hundreds-of-meters and kilometers are four different scales. But semantically, we instead might say something like "very close", "nearby", "far apart" and "very far apart".

Another approach to semantic distance is to encode the order of magnitude as an integer. This is the preferred approach herein.

For example, if we consider centimeters to be our smallest meaningful distance, we might talk about things being two centimeters apart as having a semantic distance of 1, while things a few meters apart would having a semantic distance of 2. Tens of meters would be 3, hundreds 4 and kilometers 5. We use this system as with just five integers, we can cover things on the scale of kilometers which suits most use cases. If we go even further: tens-of-kilometers and hundreds-of-kilometers becomes sufficient for almost all reasonable conversations in just 8 values (or 9 if we include a null-value).

Mathematically we do this conversion with the following formula:

$$\delta' = \lfloor \frac{log(\delta)}{log(b)} \rfloor = \lfloor log_b(\delta) \rfloor \tag{4}$$

Where...

1. δ' is the semantic distance.

2. δ is the Euclidean distance.

3. b is the basis distance (i.e. centimeters in our example above).

In Python-style pseudocode this would look like:

```
def to_semantic_distance(delta, b):
    """
        Given a Euclidean distance and a basis distance, returns
        the corresponding semantic distance.
    """
    return math.floor(math.log(delta) / math.log(b))
    # Compact alternative: return math.floor(math.log(delta, b))
```

The resulting integer distance (δ') can then be converted into arbitrary words from a list, or kept as a numerical value for easier conversion.

1.2.1 Scale Considerations with Semantic Distances

When working with semantic distances, the chosen base (b) is an important consideration. For example, if one were to use 10 as the scaling basis on a unit-length of centimeters than the system would struggle to determine the difference between 101 and 999 meters as both would be considered as a distance of 4 (i.e. $10^4\text{cm} = 100\text{m}$).

To remedy this the base used for exponentiation and units should be scaled for the target environment. For example, consider a first-person video game such as *Half-Life 2* which operates at between the centimeter to kilometer scales. The base-unit is centimeters, and the maximum unit before we lose serious fidelity concerns are kilometers (i.e. 100,000 cm). Given a target number of increments g to specify granularity over a range k we can calculate an exponentiation base with the following formula:

$$b = k^{\frac{1}{g}} \tag{5}$$

Where...

1. b is the exponent base.

2. k is the range (e.g. 100,000 cm).

3. g is the number of distance values for the system to work through the range k.

Given the previous example of $k = 100,000\text{cm}$ if we use a granularity of $g = 20$, the result would be:

$$b = 100,000^{\frac{1}{20}} = 1.7782794\ldots \tag{6}$$

From this we can determine that a nice round value around 1.75 or 1.8 will suffice to give us the granularity we're targeting. If this was used, then the semantic distance will most often range for such a system between $[0, 20]$ with only increments on orders of kilometers going beyond 20.

1.3 A Full View of Semantic Vectors

With this understanding of semantic direction and distance, we can now understand that a semantic vector just stores the general direction of something (i.e. North/South, East/West and Up/Down) and the order of magnitudes of distance (i.e. 1cm, 100 cm, etc.) to encode a vague approximation of a Euclidean Vector in a meaningful manner.

These vectors have the benefits of being both semantically rich, and having limited mathematical domains over which they're defined.

1.4 Combining Semantic Vectors

It's often useful to combine two semantic vectors. To do this, they're converted back into Euclidean Vectors, added together, then converted back to Semantic Vectors. Conversion is done by converting the semantic-direction into a Euclidean direction vector, and multiplying it by the result of converting a given semantic-distance back into a distance value.

Mathematically, given a semantic vector $A = (N, \Delta)$ conversion back to a Euclidean vector would look like:

1. Convert the semantic direction N to the corresponding Euclidean basis-vector ($\mathbf{B}_N$).
2. Convert the semantic distance Δ into a Euclidean distance d via the formula $d = b^{\Delta}$ where b is the basis-magnitude for distance.
3. Multiply the distance by the basis vector (i.e. $\mathbf{v} = \mathbf{B}_N \cdot d$) to get the Euclidean vector approximation of the semantic vector ($\mathbf{v}$).

Once two Euclidean approximation vectors ($\mathbf{v}_1, \mathbf{v}_2$) are produced, they're then added together.

$$\mathbf{v}_3 = \mathbf{v}_1 + \mathbf{v}_2 \tag{7}$$

And then the resulting vector $\mathbf{v}_3$ is converted back into a semantic vector using the previously described methods.

This method is employed because when combining semantic vectors, the magnitudes of corresponding distances relate to the manner in which the vectors themselves change direction. For example, if you combine the ideas of going a few meters west, and a kilometer north-east, you would still just say "north-east".

1.4.1 Caveat in Semantic-Euclidean Conversion

It's worth noting that because semantic-vectors store a basis-vector (e.g. $(1, 0, 1)$) instead of a normalized vector (e.g. $(0.8, 0, 0.2)$) they "lose" information, and conversion back into Euclidean vectors, combination and conversion back can cause a margin of error to occur in some cases.

To alleviate this, at the cost of increased memory usage, an implementation can store the normalized Euclidean vector in the semantic vector as well.

1.5 Conclusion on Semantic Vectors

In review, semantic vectors are just a data structure that looks like:

```
{"direction": "NW", "distance": 3} // 3 = a few meters.
```

formalized in a more rigorous mathematical way. They're used with specific systems, and generally represent more "human"-like representations of information than the raw Euclidean vectors.