What you'll be able to do!

machine translation "hello!" → "bonjour!"

document search "Can I get a refund?" → "What's your return policy?" ... "May I get my money back?"

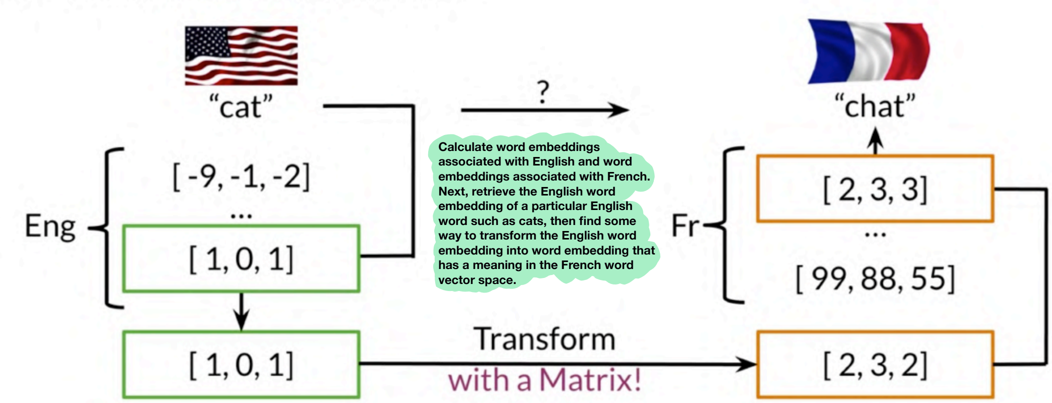
Learning Objectives

- Transform vector
- "K nearest neighbors"
- Hash tables
- Divide vector space into regions
- Locality sensitive hashing
- Approximated nearest neighbors

→ Machine translation

→ Document search

Overview of Translation



Transforming vectors

```
R = np.array([[2,0],
              [0,-2]])
x = np.array([[1,1]])
np.dot(x,R)
```

```
array([[2,-2]])
```

Align word vectors

Minimize distance

$$\mathbf{XR} \approx \mathbf{Y}$$

$$\underset{\mathbf{X}}{\begin{pmatrix} [\text{“cat” vector}] \\ [\text{... vector}] \\ [\text{“zebra” vector}] \end{pmatrix}} \qquad \underset{\mathbf{Y}}{\begin{pmatrix} [\text{“chat” vecteur}] \\ [\text{... vecteur}] \\ [\text{“zébresse” vecteur}] \end{pmatrix}}$$

subsets of the full vocabulary

Solving for R

initialize R

in a loop:

English, Parameters, French

$$Loss = \| \underset{\text{English}}{\mathbf{X}}\underset{\text{Parameters}}{\mathbf{R}} - \underset{\text{French}}{\mathbf{Y}} \|_F$$

$$g = \frac{d}{dR} Loss \qquad \text{gradient}$$

$$R = R - \alpha g \qquad \text{update}$$

Frobenius norm

$$\| \mathbf{XR} - \mathbf{Y} \|_F$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

$$\|\mathbf{A}_F\| = \sqrt{2^2 + 2^2 + 2^2 + 2^2}$$

$$\|\mathbf{A}_F\| = 4$$

$$\|\mathbf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

Square all elements in A and sum them

Frobenius norm

```
A = np.array([[2,2],
              [2,2]])

A_squared = np.square(A)
A_squared
```

```
array([[4,4],
       [4,4]])
```

```
A_Frobenious = np.sqrt(np.sum(A_squared))
A_Frobenious
```

```
4.0
```

Frobenius norm squared

$$\|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$\mathbf{A} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}$$

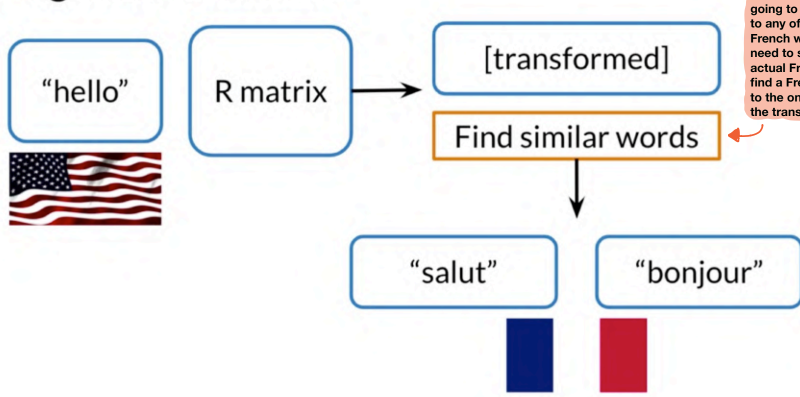
$$\|\mathbf{A}\|_F^2 = \left(\sqrt{2^2 + 2^2 + 2^2 + 2^2}\right)^2$$

Gradient

$$Loss = \|\mathbf{XR} - \mathbf{Y}\|_F^2$$

$$g = \frac{d}{dR} Loss = \frac{2}{m} \left(\mathbf{X}^T(\mathbf{XR} - \mathbf{Y})\right)$$

Finding the translation



Notice that it transformed word vector after the transformation of its embedding through an R matrix would be in the French word vector space. But it is not going to be necessarily identical to any of the word vectors in the French word vector space. You need to search through the actual French word vectors to find a French word that is similar to the one that you created from the transformation.

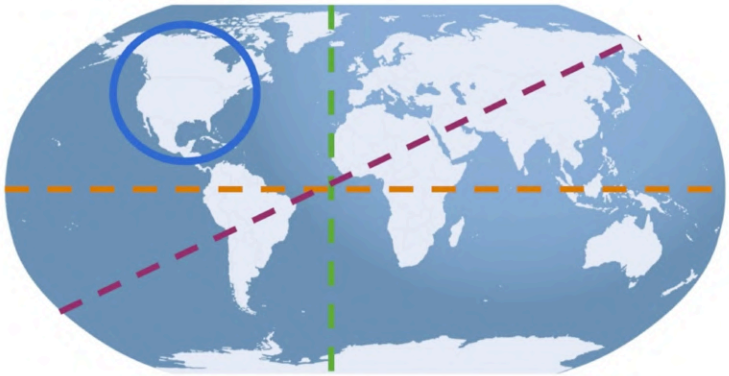
Nearest neighbours

You

San Francisco

Friend	Location	Nearest
	Shanghai	2
	Bangalore	3
	Los Angeles	1

Nearest neighbors

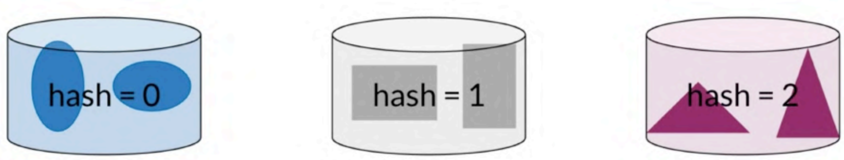


Hash tables!

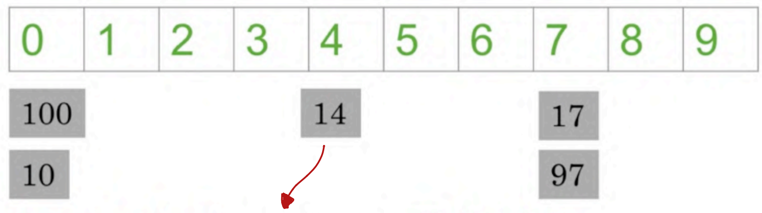
Summary

- K-nearest neighbors, for closest matches
- Hash tables

Hash tables



Hash function



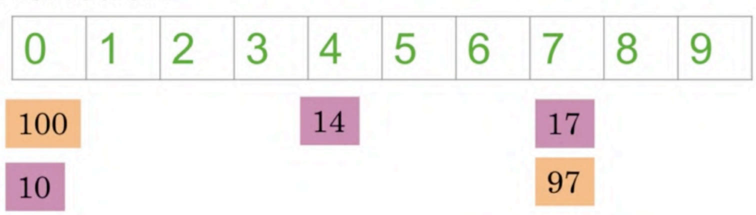
Hash function (vector) ⟶ Hash value

Hash value = vector % number of buckets

Create a basic hash table

```python
def basic_hash_table(value_l,n_buckets):

    def hash_function(value_l,n_buckets):
        return int(value) % n_buckets

    hash_table = {i:[] for i in range(n_buckets)}
    for value in value_l:
        hash_value = hash_function(value,n_buckets)
        hash_table[hash_value].append(value)

    return hash_table
```

Hash function

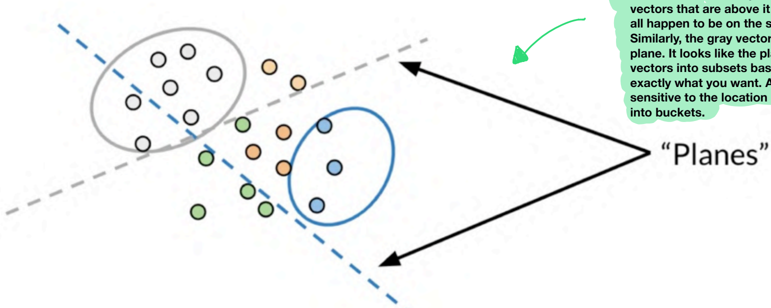


Hash function by location?

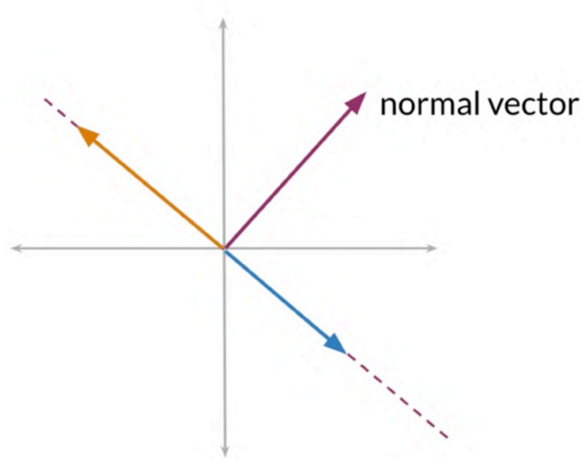
Locality sensitive hashing, next!

Locality Sensitive Hashing

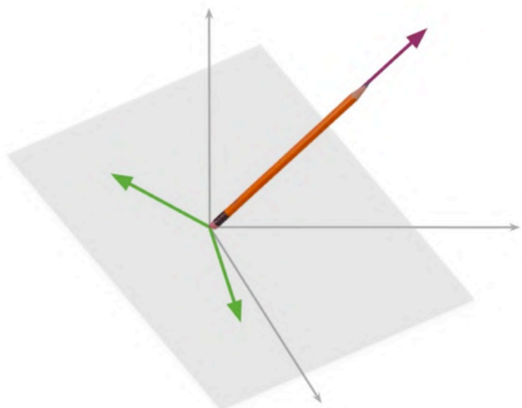
Let’s say you want to find a way to know that these blue dots are somehow close to each other, and that these gray dots are also related to each other. First, divide the space using these dashed lines, which I'll call planes. I'll explain why I called them planes in a bit. Notice how the blue plane slices up the space into vectors that are above it or below it. The blue vectors all happen to be on the same side of the blue plane. Similarly, the gray vectors happen to be above the gray plane. It looks like the planes can help us bucket the vectors into subsets based on their location. This is exactly what you want. A hashing function that is sensitive to the location of the items that it's assigning into buckets.



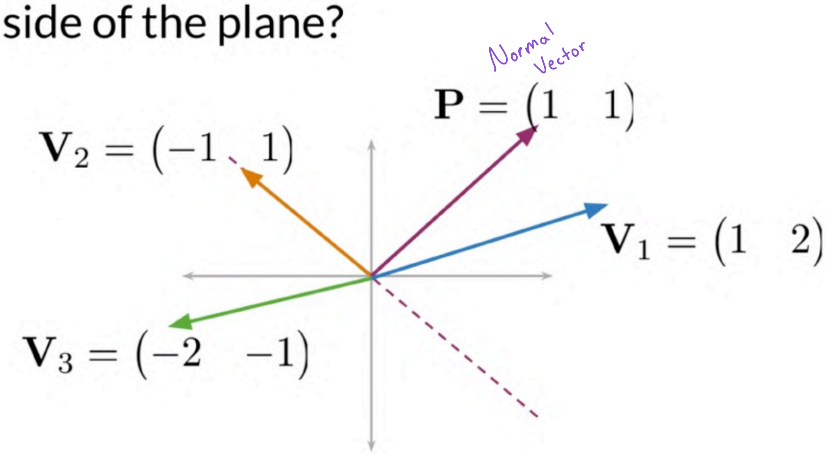
Planes



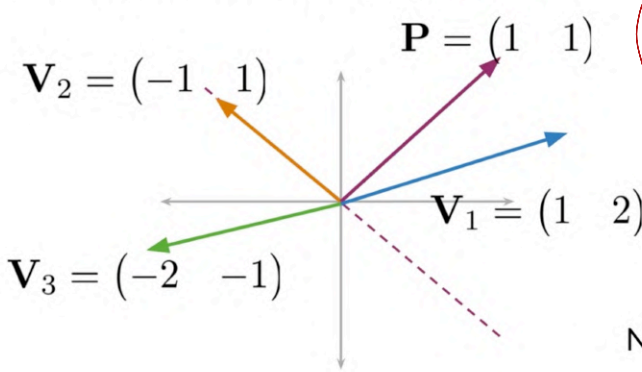
Planes



Which side of the plane?



Which side of the plane?



Do you notice something about the signs and how they're related to their position relative to the purple plane?

- Dot product is positive:
  - The vector is on one side of the plane.
- Dot product is negative
  - The vector is on the opposite side of the plane.
- Dot product is zero
  - The vector is on the plane.

$$\mathbf{P}\mathbf{V}_1^T = 3$$

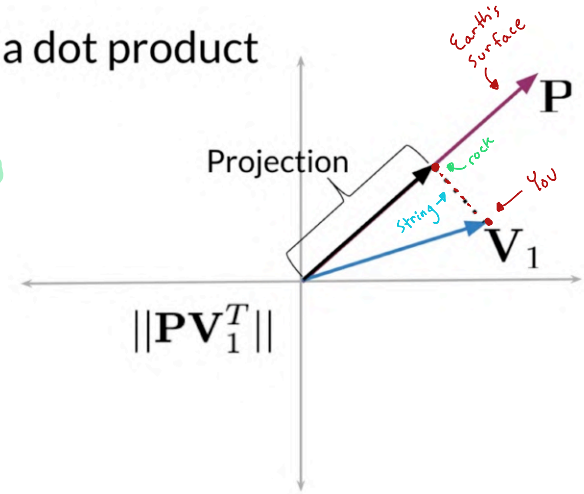
$$\mathbf{P}\mathbf{V}_2^T = 0$$

$$\mathbf{P}\mathbf{V}_3^T = -3$$

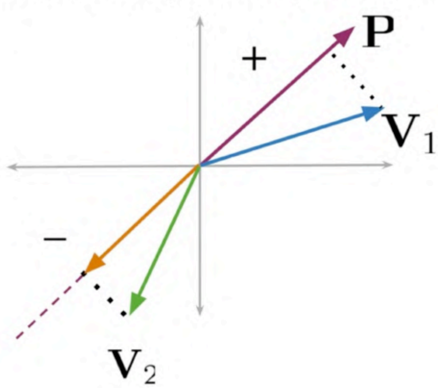
Notice the signs?

Visualizing a dot product

To visualize the dot product, imagine one of the vectors such as P, as if it's the surface of the Earth. Gravity pulls all objects straight down towards the surface of the Earth. Next, pretend you're standing at the end of the vector, V1. You tie a string to a rock and let gravity pull the rock to the surface of vector P. The string is perpendicular to vector P. Now, if you draw a vector that's in the same direction of P but ends up at the rock, you'll have what's called the projection of vector V1 onto vector P. The magnitude or length of that vector is equal to the dot product of V1 and P.



Visualizing a dot product



Sign indicates direction

Furthermore, if you had this other green vector and projected it onto vector P, the projected vector would be pointing in the parallel but opposite direction of P. The dot product would be a negative number. This means that the sign of the dot product indicates the direction of the projection with respect to the purple normal vector. So whether the dot product is positive or negative can tell you whether the vector V1 or V2 are on one side of the plane or the other. Let's use code to check which side of the plane the vector is on. The function side of plane takes in the normal vector P, and the vector v. Use numpy dot to take the dot products. Use numpy.sign to get a plus one if the dot product is positive, minus one if the dot product is negative, or zero if the dot product is zero. I'm using numpy.asscalar. Notice the pronunciation of that function. If a vector can be represented as a single scalar, this function retrieves that scalar, and that's it.

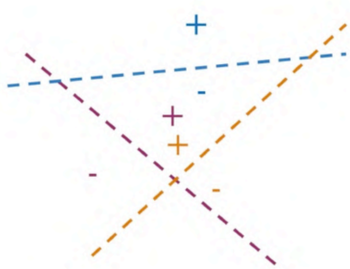
Which side of the plane?

```
def side_of_plane(P,v):
    dotproduct = np.dot(P,v.T)
    sign_of_dot_product = np.sign(dotproduct)
    sign_of_dot_product_scalar= np.asscalar(sign_of_dot_product)
    return sign_of_dot_product_scalar
```

Outline

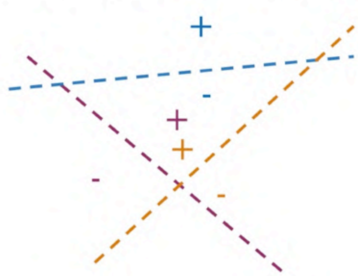
- Multiple planes ⟶ Dot products ⟶ Hash values

Multiple planes



⟶ single hash value?

Multiple planes, single hash value?



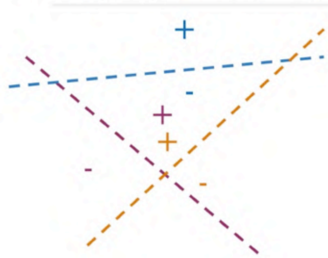
$$\boxed{\mathbf{P}_1\mathbf{v}^T = 3, sign_1 = +1, h_1 = 1}$$

$$\boxed{\mathbf{P}_2\mathbf{v}^T = 5, sign_2 = +1, h_2 = 1}$$

$$\boxed{\mathbf{P}_3\mathbf{v}^T = -2, sign_3 = -1, h_3 = 0}$$

$$hash = 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3$$
$$= 1 \times 1 + 2 \times 1 + 4 \times 0$$
$$= 3$$

Multiple planes, single hash value!



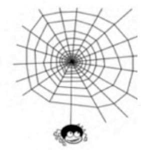
$$sign_i \geq 0, \rightarrow h_i = 1$$
$$sign_i < 0, \rightarrow h_i = 0$$

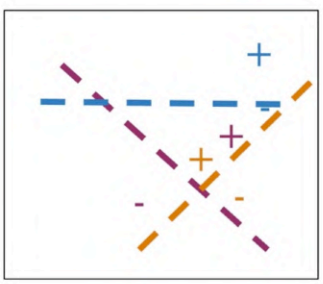
$$\text{hash} = \sum_{i}^{H} 2^i \times h_i$$

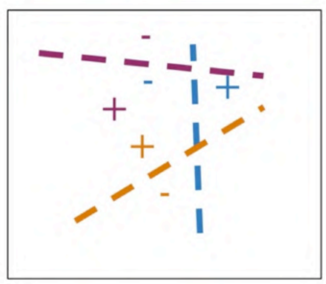
Multiple planes, single hash value!!

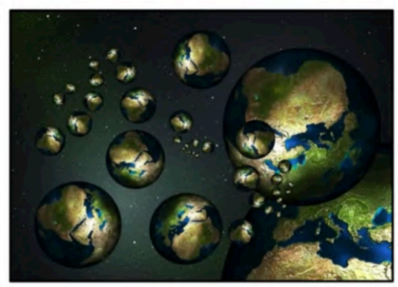
```python
def hash_multiple_plane(P_l,v):

    hash_value = 0

    for i, P in enumerate(P_l):
        sign = side_of_plane(P,v)
        hash_i = 1 if sign >=0 else 0
        hash_value += 2**i * hash_i

    return hash_value
```

Random planes



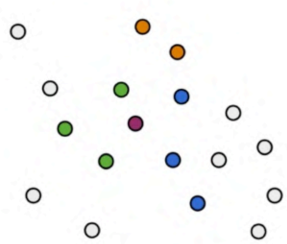






Cultural reference: Spider-Man: Into the Spider-Verse

Multiple sets of random planes



Approximate nearest (friendly) neighbors

Make one set of random planes

```python
num_dimensions = 2 #300 in assignment
num_planes = 3 #10 in assignment

random_planes_matrix = np.random.normal(
                       size=(num_planes,
                             num_dimensions))
```

```
array([[ 1.76405235  0.40015721]
       [ 0.97873798  2.2408932 ]
       [ 1.86755799 -0.97727788]])
```

```python
v = np.array([[2,2]])
```

```python
def side_of_plane_matrix(P,v):
    dotproduct = np.dot(P,v.T)
    sign_of_dot_product = np.sign(dotproduct)
    return sign_of_dot_product

num_planes_matrix = side_of_plane_matrix(
                    random_planes_matrix,v)
```

```
array([[1.]
       [1.]
       [1.]])
```

See notebook for calculating the hash value!

Document representation

I love learning!	[?, ?, ?]
I	[1, 0, 1]
	+
love	[-1, 0, 1]
	+
learning	[1, 0, 1]
	=
I love learning!	[1, 0, 3]

word vector

Add all word vectors

Document Search

Document vectors

```python
word_embedding = {"I": np.array([1,0,1]),
                  "love": np.array([-1,0,1]),
                  "learning": np.array([1,0,1])}

words_in_document = ['I', 'love', 'learning']

document_embedding = np.array([0,0,0])

for word in words_in_document:
    document_embedding += word_embedding.get(word,0)

print(document_embedding)
```

initialize

```
array([1 0 3])
```