



deeplearning.ai

Overview

What you'll be able to do!

machine translation

"hello!"



"bonjour!"

document search

"Can I get a
refund?"



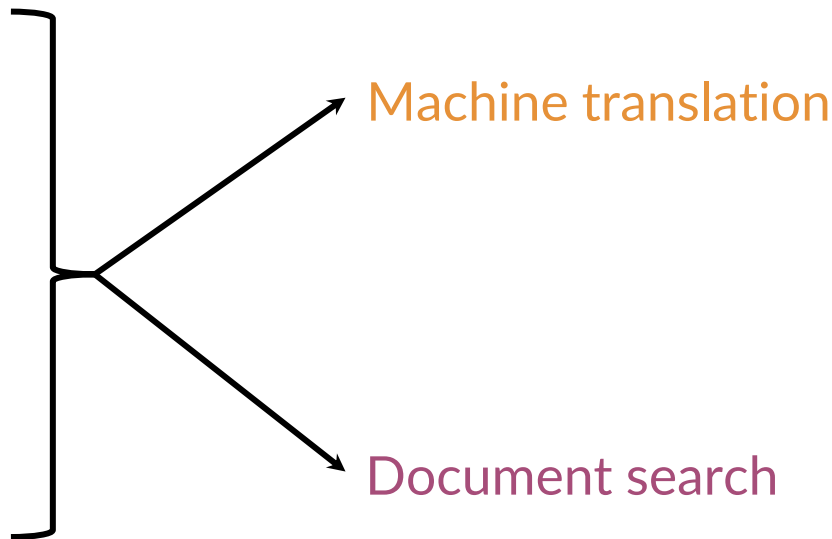
"What's your return
policy?"

...

"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





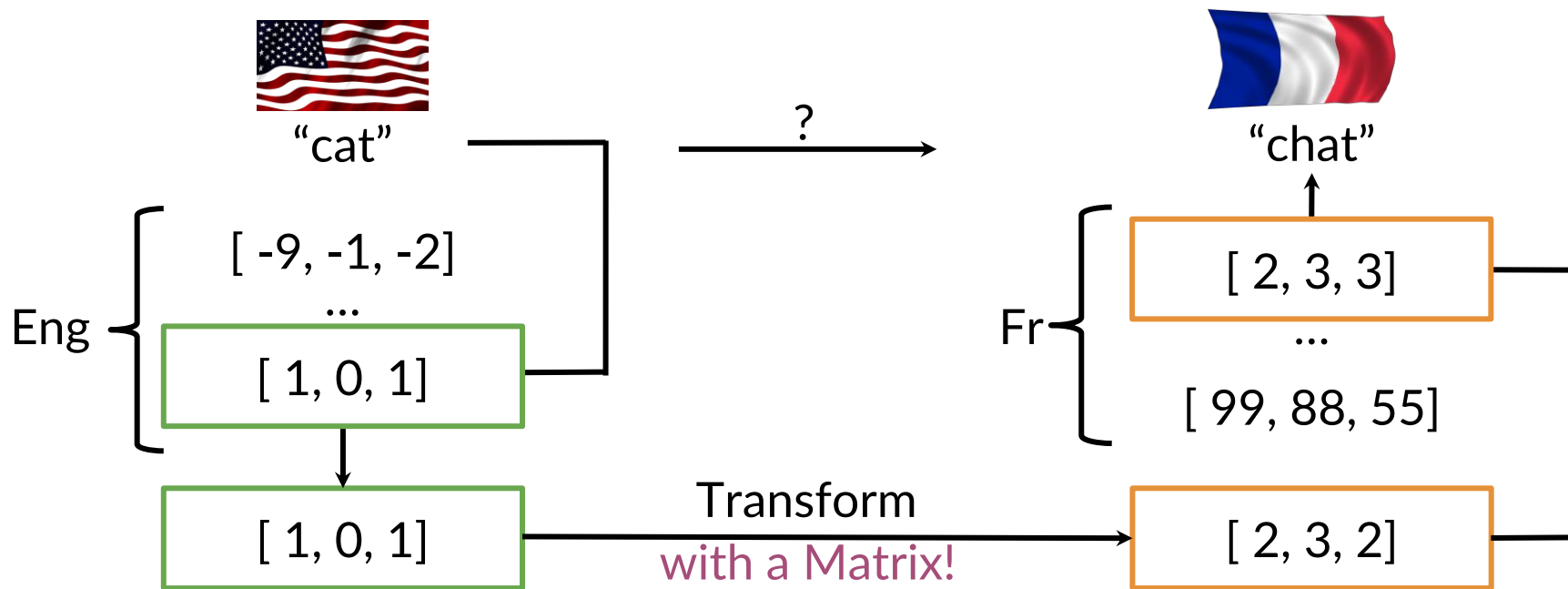
deeplearning.ai

Transforming word vectors

Outline

- Translation = Transformation
- How to get a good transformation

Overview of Translation



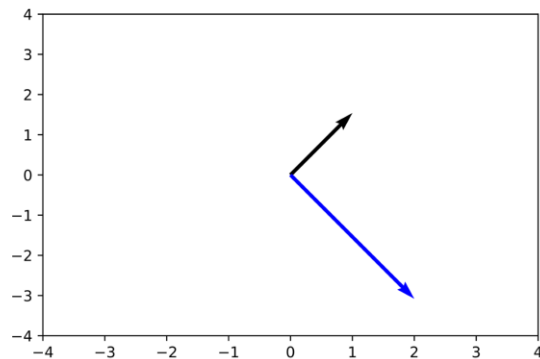
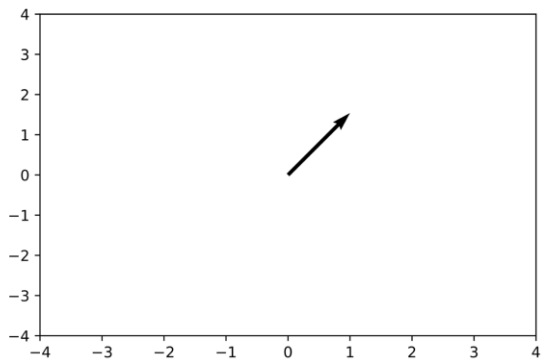
Transforming vectors

$$\begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix} = \begin{pmatrix} 2 & -2 \end{pmatrix}$$

X

R

Y



Transforming vectors

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

Try it
yourself!

Align word vectors

$$\begin{pmatrix} [\text{"cat" vector}] \\ [\dots \text{vector}] \\ [\text{"zebra" vector}] \end{pmatrix} \mathbf{X} \mathbf{R} \approx \mathbf{Y} \begin{pmatrix} [\text{"chat" vecteur}] \\ [\dots \text{vecteur}] \\ [\text{"z bresse" vecteur}] \end{pmatrix}$$

\mathbf{X} \mathbf{Y}

subsets of the full
vocabulary

Solving for R

initialize R

in a loop:

$$Loss = \| \mathbf{X}\mathbf{R} - \mathbf{Y} \|_F$$

$$g = \frac{d}{dR} Loss$$

gradient

$$R = R - \alpha g$$

update

Frobenius norm

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

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

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

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

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

Frobenius norm

```
A = np.array([[2,2],  
- - -  
A_squared = np.square(A)  
A_squared  
array([[4,4],
```

Try it yourself!

```
A_Frobenious = np.sqrt(np.sum(A_squared))  
A_Frobenious  
4.0
```

Frobenius norm squared

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

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

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

$$\|\mathbf{A}\|_F^2 = 16$$

Gradient

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

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

Implement in the assignment!

Summary

- $\mathbf{X}\mathbf{R} \approx \mathbf{Y}$
- minimize $\|\mathbf{X}\mathbf{R} - \mathbf{Y}\|_F^2$



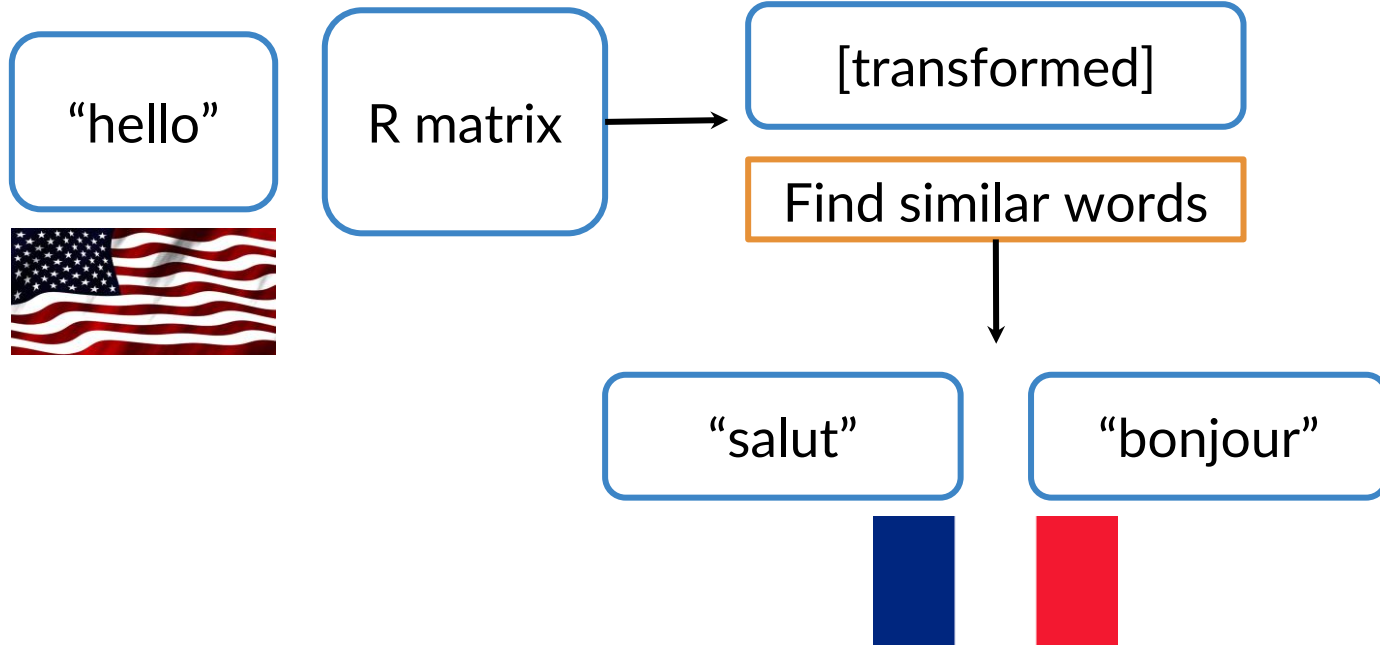
deeplearning.ai

K-nearest neighbors

Outline

- K closest matches → K-nearest neighbors

Finding the translation



Nearest neighbours



You

San Francisco

Friend



Location

Shanghai

Nearest

2



Bangalore

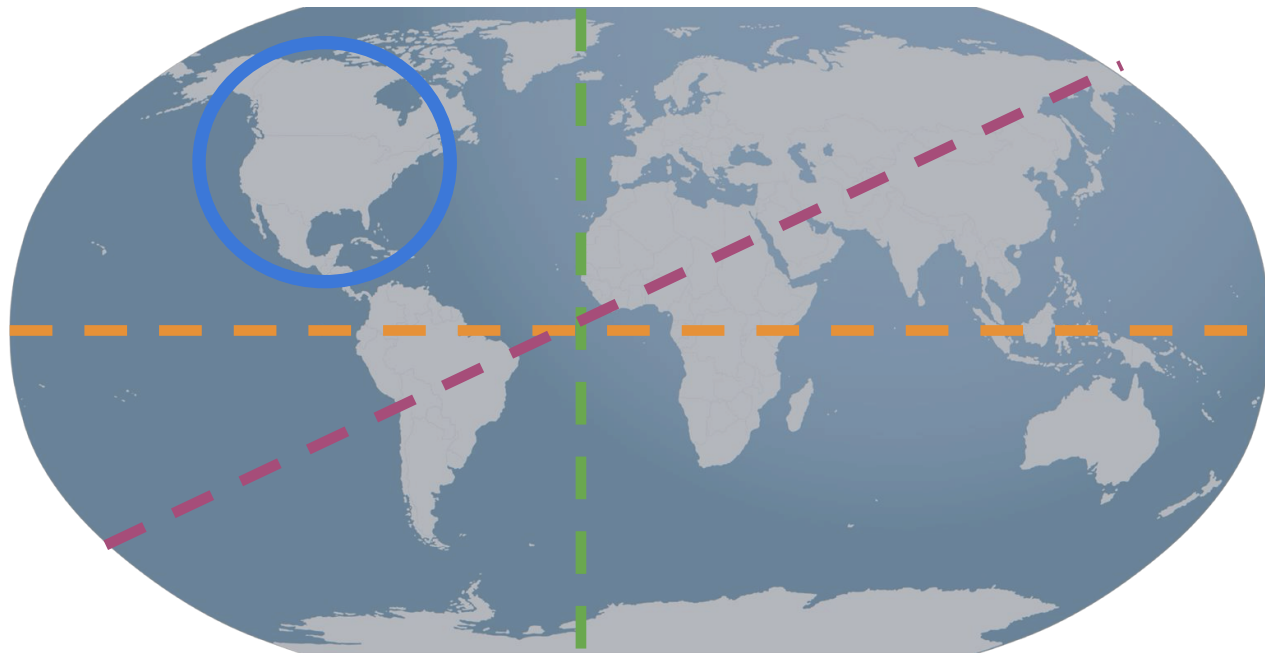
3



Los Angeles

1

Nearest neighbors



Hash
tables!



Summary

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



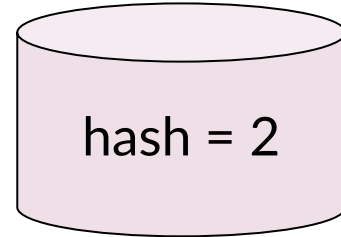
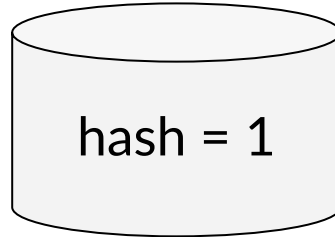
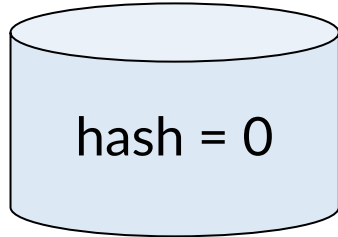
deeplearning.ai

Hash tables and hash functions

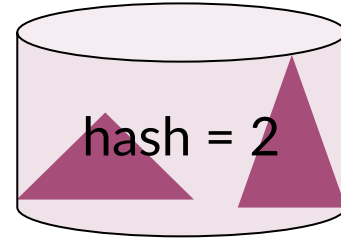
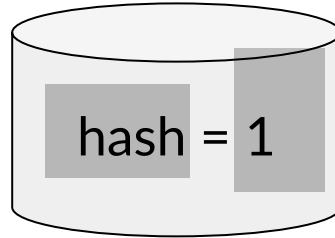
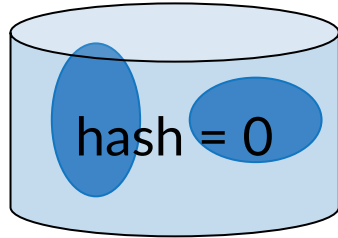
Outline

- Hash values
- Hash functions

Hash tables



Hash tables



Hash function

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

100

14

17

10

97

Hash function (vector) \longrightarrow Hash value

Hash value = vector % number of buckets

Create a basic hash table

```
def basic_hash_table(value_l, n_buckets):  
    def hash_function(value_l, n_buckets):  
        return int(value_l) % n_buckets  
    hash_table = {i:[] for i in range(n_buckets)}  
    for value in value_l:  
        hash_value =  
hash_function(value, n_buckets)  
        return hash_table
```

Hash function

0	1	2	3	4	5	6	7	8	9
100				14			17		
10							97		

Hash function by location?

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

14

100

10

97

17

Locality sensitive hashing, next!

Summary

Hash function (vector) \longrightarrow Hash value



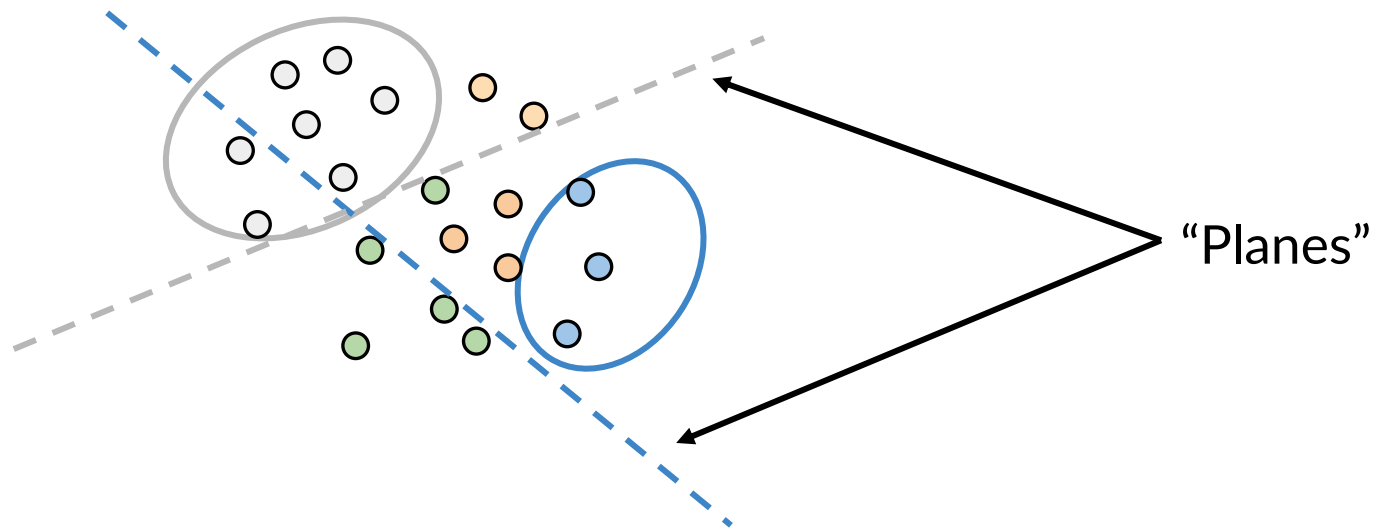
deeplearning.ai

Locality sensitive hashing

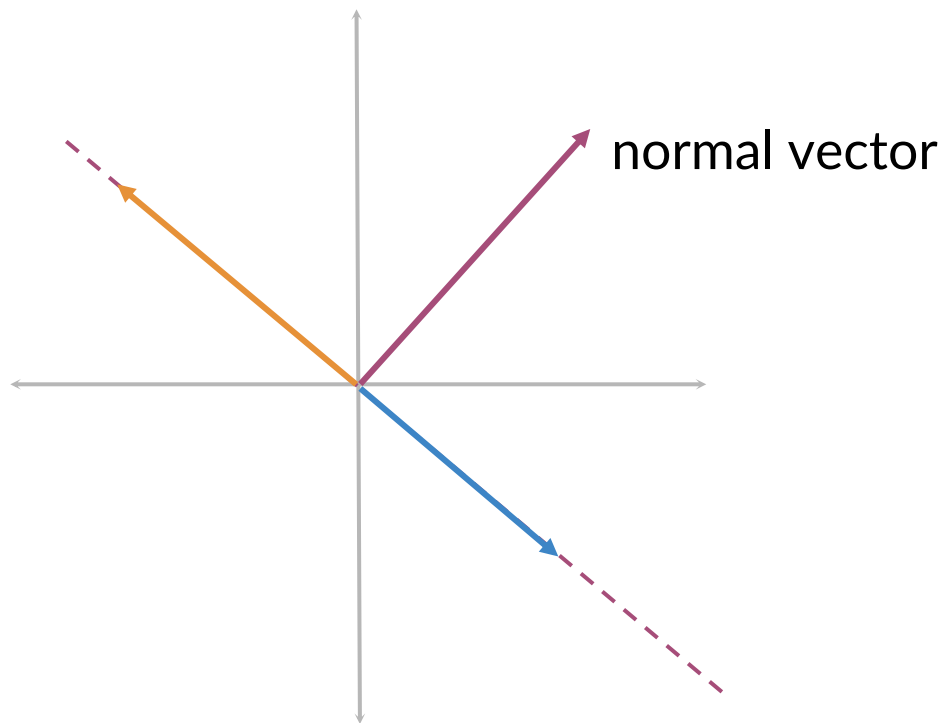
Outline

- Locality sensitive hashing with planes in vector spaces

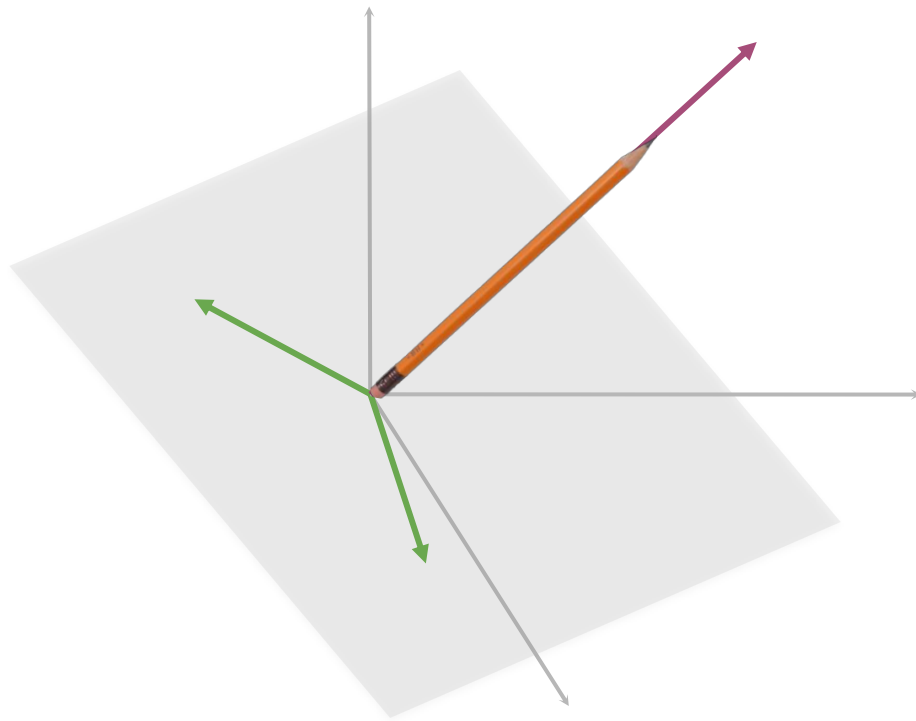
Locality Sensitive Hashing



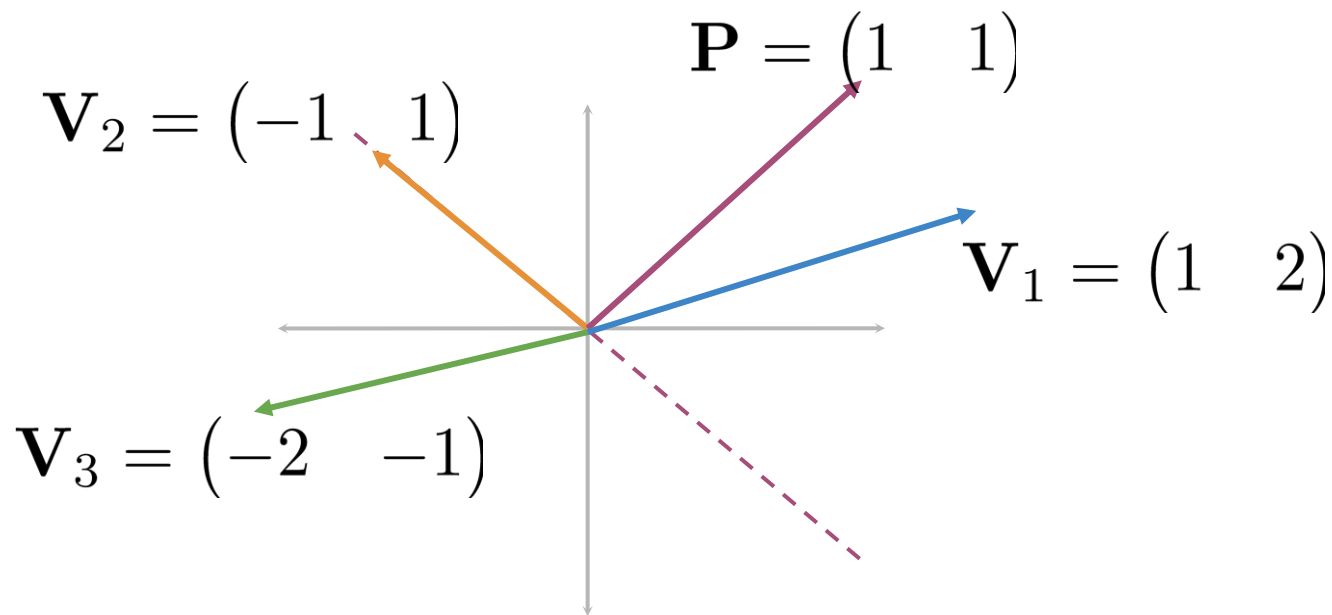
Planes



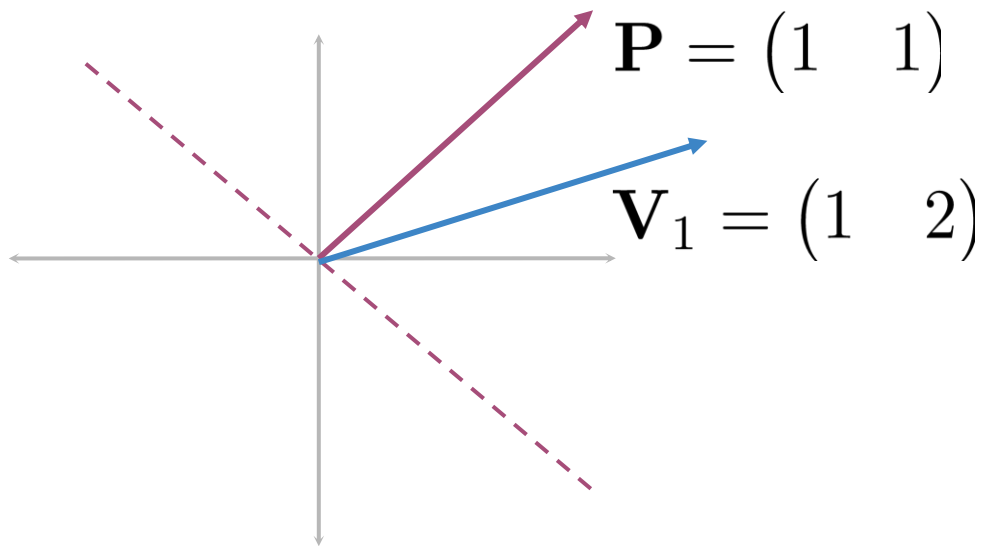
Planes



Which side of the plane?



Which side of the plane?

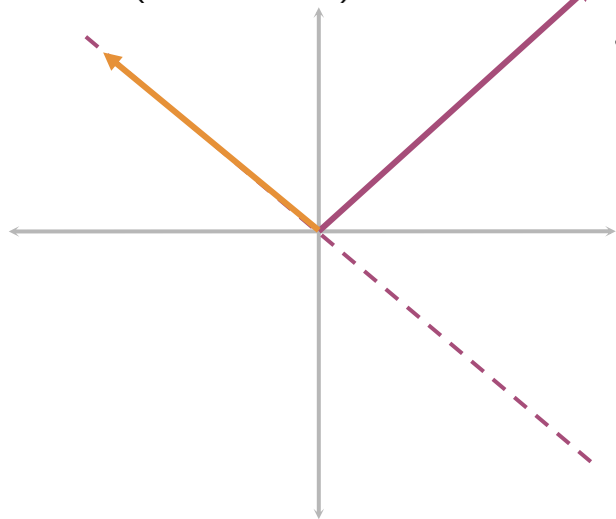


$$\mathbf{P}\mathbf{V}_1^T = 3$$

Which side of the plane?

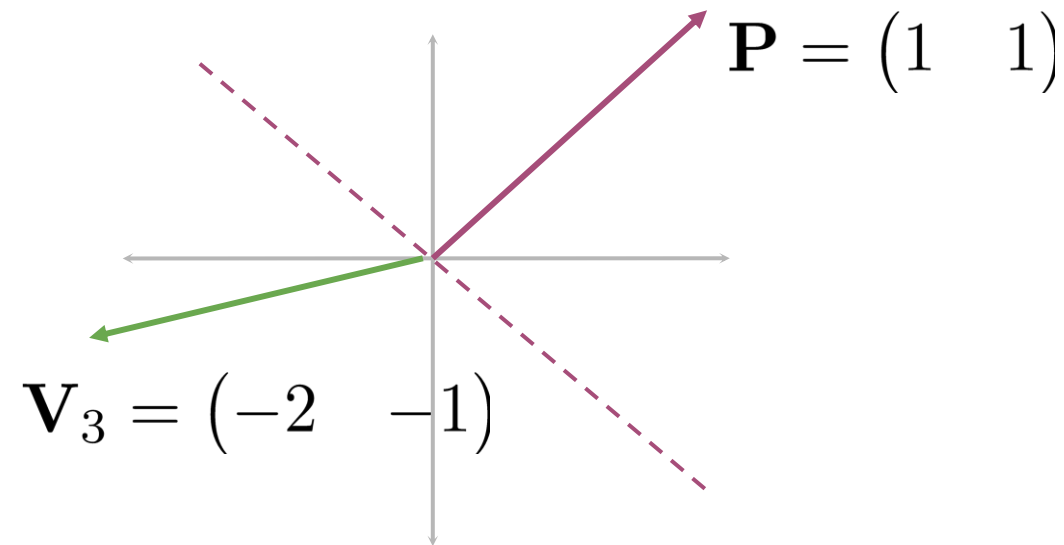
$$\mathbf{V}_2 = \begin{pmatrix} -1 & 1 \end{pmatrix}$$

$$\mathbf{P} = \begin{pmatrix} 1 & 1 \end{pmatrix}$$



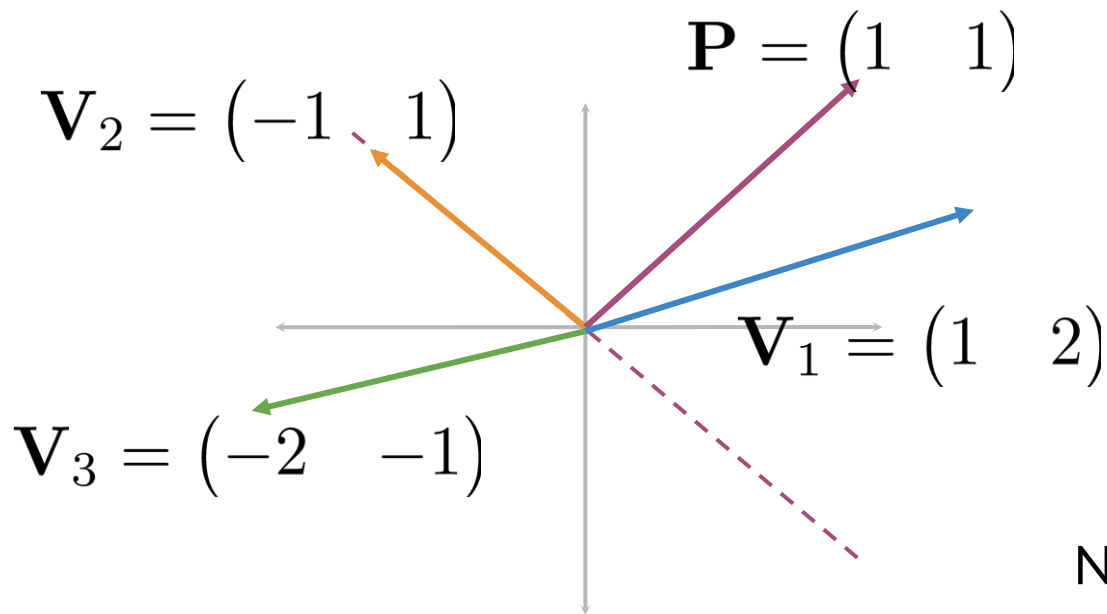
$$\mathbf{P}\mathbf{V}_2^T = 0$$

Which side of the plane?



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

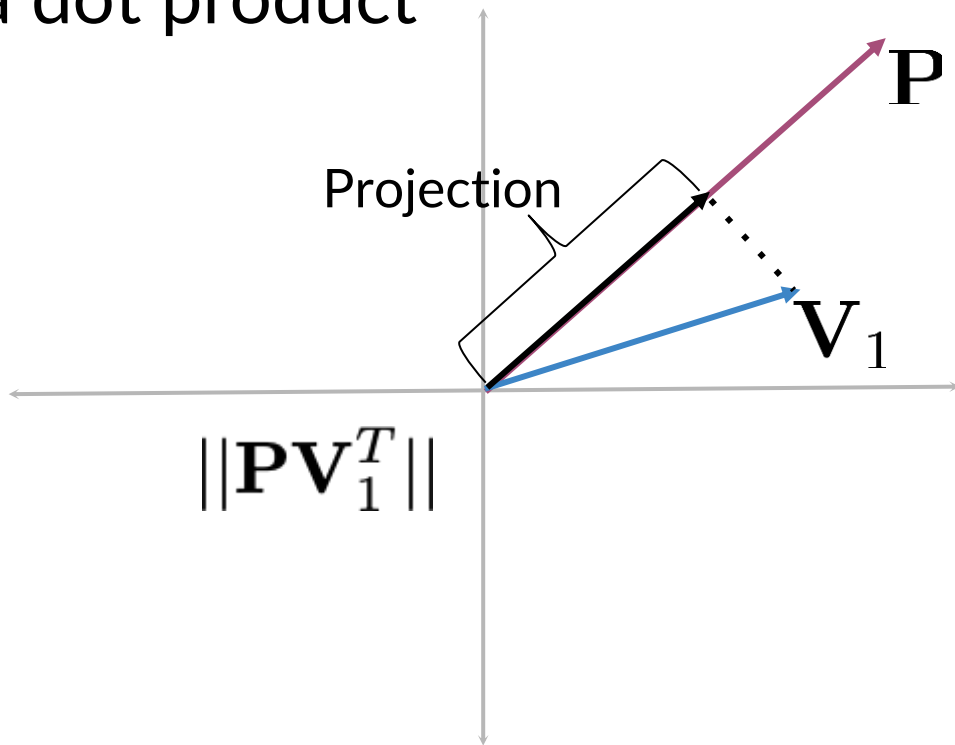
Which side of the plane?



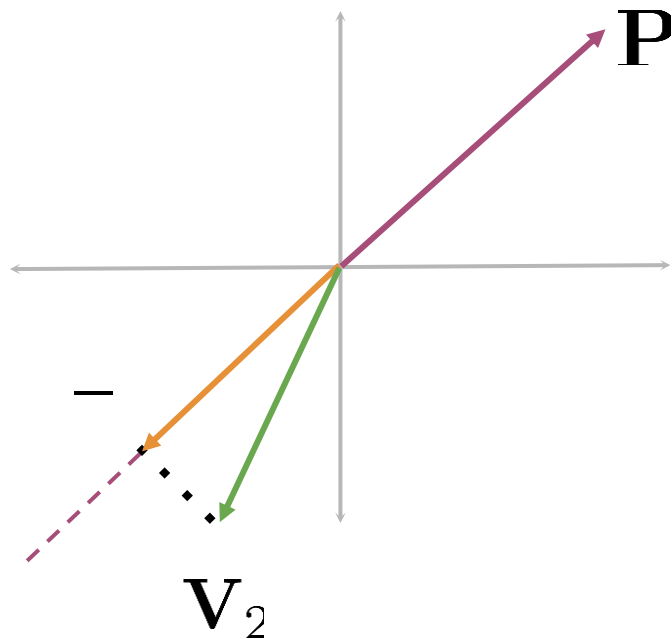
$$\begin{aligned}\mathbf{P}\mathbf{V}_1^T &= 3 \\ \mathbf{P}\mathbf{V}_2^T &= 0 \\ \mathbf{P}\mathbf{V}_3^T &= -3\end{aligned}$$

Notice the signs?

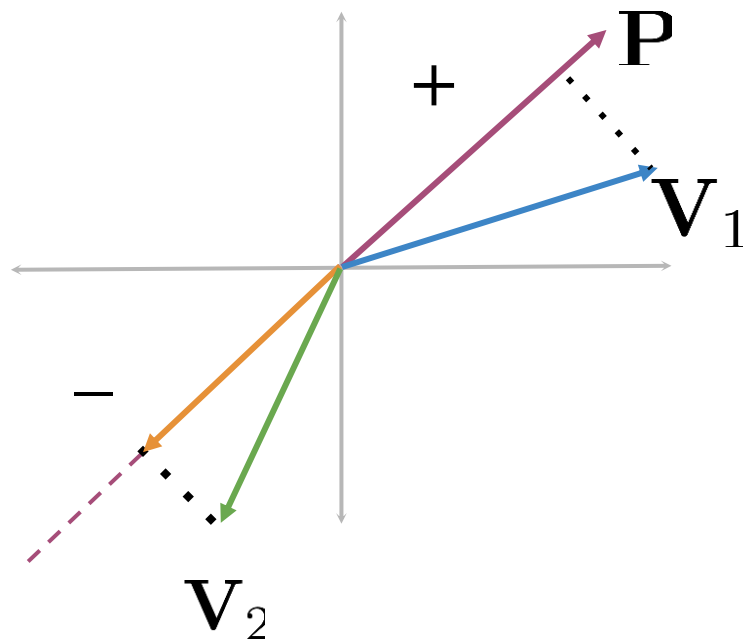
Visualizing a dot product



Visualizing a dot product



Visualizing a dot product



Sign indicates direction

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
```

Try it!

Summary

- Sign of dot product \longrightarrow Hash values



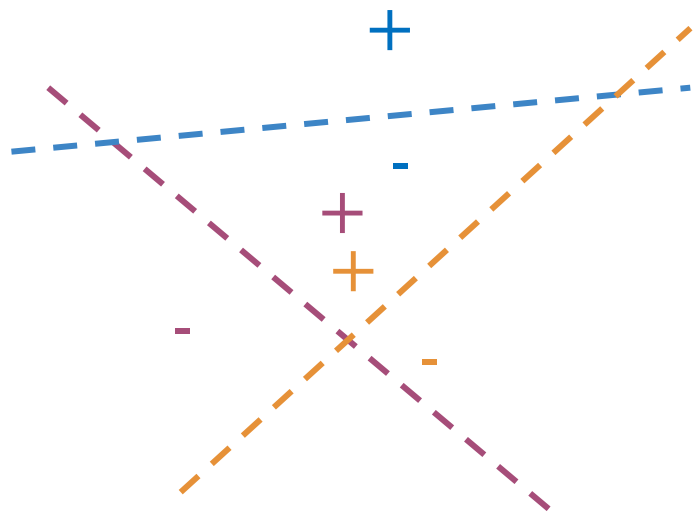
deeplearning.ai

Multiple Planes

Outline

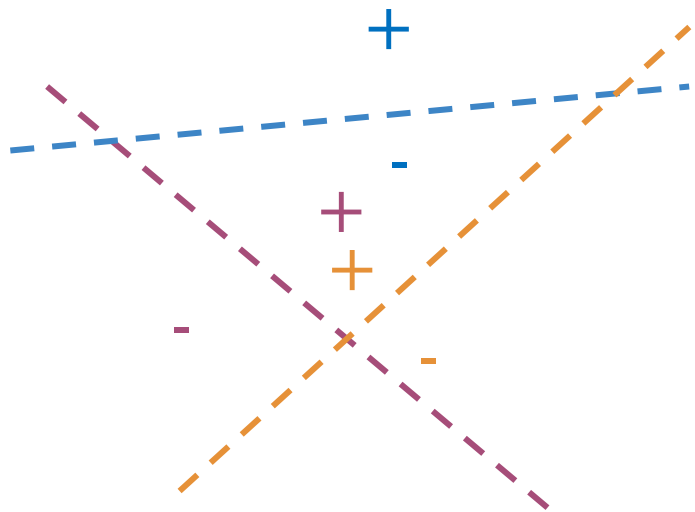
- Multiple planes \longrightarrow Dot products \longrightarrow Hash values

Multiple planes



single hash value?

Multiple planes, single hash value?



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

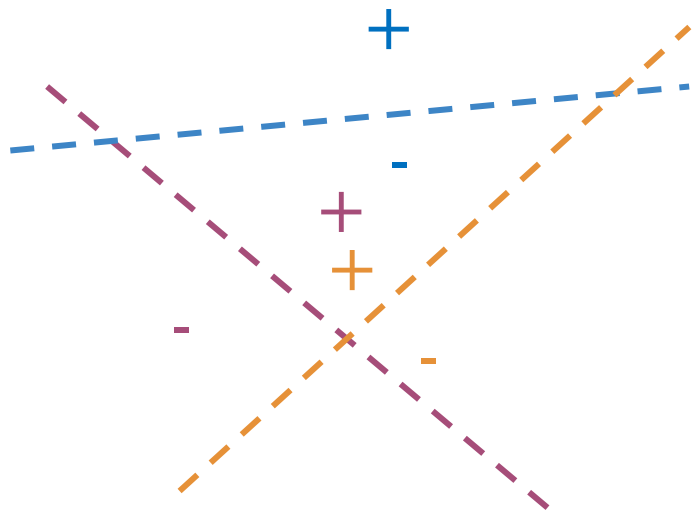
$$\mathbf{P}_2 \mathbf{v}^T = 5, \text{sign}_2 = +1, h_2 = 1$$

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

$$\begin{aligned} \text{hash} &= 2^0 \times h_1 + 2^1 \times h_2 + 2^2 \times h_3 \\ &= 1 \times 1 + 2 \times 1 + 4 \times 0 \end{aligned}$$

$$= 3$$

Multiple planes, single hash value!



$$\text{sign}_i \geq 0, \rightarrow h_i = 1$$

$$\text{sign}_i < 0, \rightarrow h_i = 0$$

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

Multiple planes, single hash value!!

```
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
```

Try it!

Summary





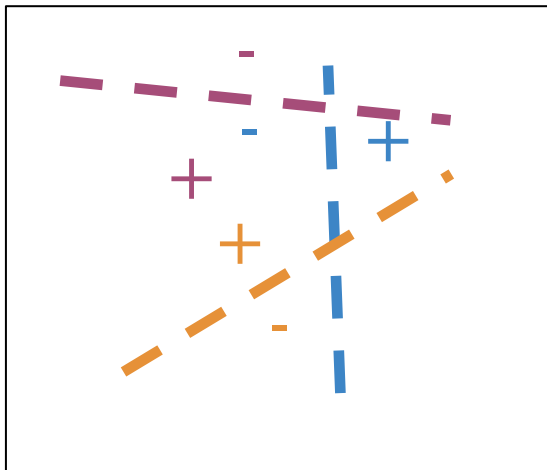
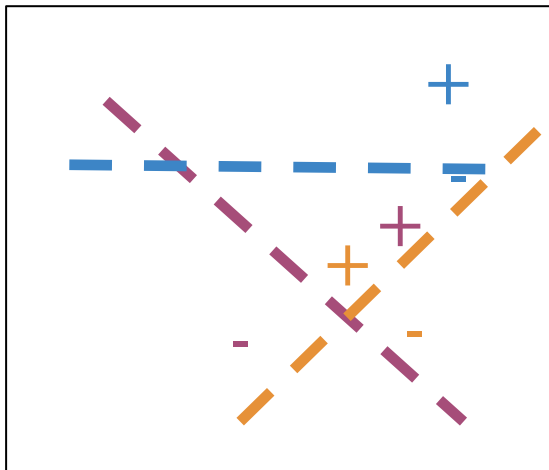
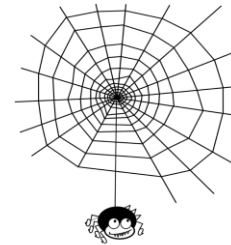
deeplearning.ai

Approximate nearest neighbors

Outline

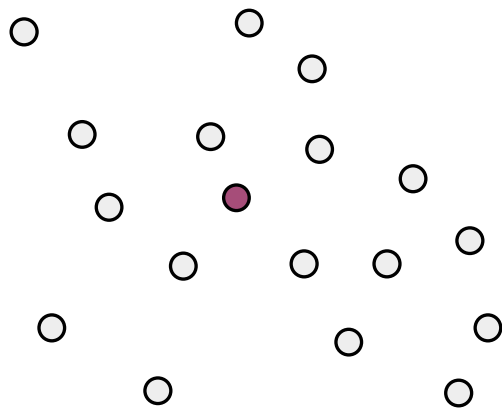
- Multiple sets of planes for approximate K-nearest neighbors

Random planes

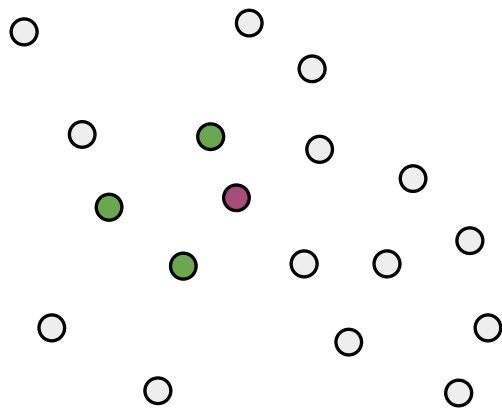


Cultural reference: Spider-Man: Into the Spider-Verse

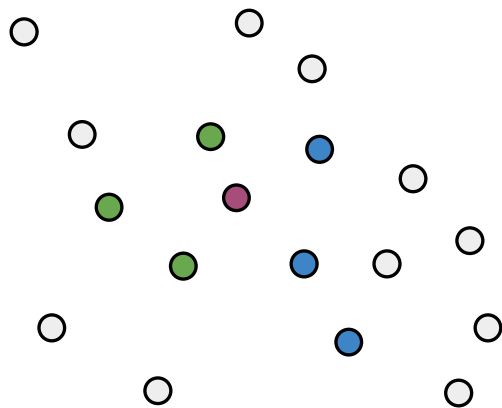
Multiple sets of random planes



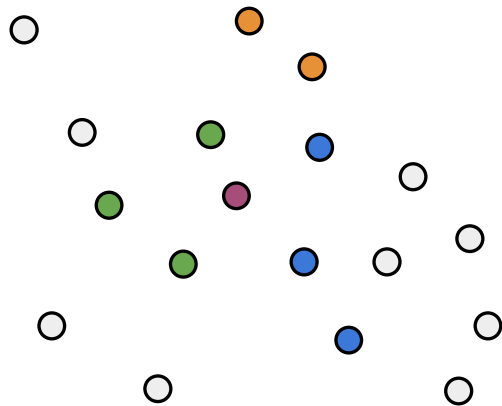
Multiple sets of random planes



Multiple sets of random planes



Multiple sets of random planes



Approximate nearest
(friendly) neighbors

Make one set of random planes

```
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]])
```

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

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

num_planes_matrix = side_of_plane_matrix(
    random_planes_matrix,v)
```

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

See notebook for calculating the hash value!

Summary

- Multiple universes \longrightarrow Locality sensitive hashing \longrightarrow A. K-NN



deeplearning.ai

Searching documents

Outline

- Representation for documents
- Document search with K-nearest neighbors

Document representation

I love learning!

[?, ?, ?]

I

[1, 0, 1]

love

[-1, 0, 1]

learning

[1, 0, 1]

I love learning!

[1, 0, 3]

Document Search

K-NN!

Document vectors

```
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[word]

print(document_embedding)

array([1 0 3])
```

Try it!

Revisit Learning Objectives

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

