实验2-向量

大纲

创建向量

标量和向量

标量就是一个简单的数，例如：$weight = 3$，$speed = 5$，$price = 7.5$。

In [4]:
```
weight = 3
speed = 5
price = 7.5

# 这些数值都是标量
print(f'weight: {weight}, speed: {speed}, price: {price}')
```

```
weight: 3, speed: 5, price: 7.5
```

向量是一系列的有顺序的数，例如：$v = [1, 2, 3, 4]$，$w = [2, 4, 6, 8]$。在Python中，Numpy库是最常见的用来处理向量的库。

In [5]:
```
# 安装numpy
%pip install numpy
```

```
Requirement already satisfied: numpy in c:\python312\lib\site-packages (1.26.1)Not
e: you may need to restart the kernel to use updated packages.
```

In [6]:
```
# 导入numpy
import numpy as np

# 创建numpy向量
vector1 = np.array([1, 2, 3, 4])
vector2 = np.array([5, 6, 7, 8])

# 打印向量
print(f'vector1: {vector1}')
print(f'vector2: {vector2}')
```

```
vector1: [1 2 3 4]
vector2: [5 6 7 8]
```

In [7]:
```
# 将标量放入到numpy向量中
weight = 3
speed = 5
price = 7.5
car1 = np.array([weight, speed, price])
print(f'car1: {car1}')
```

```
car1: [3.  5.  7.5]
```

In [14]:
```
# 向量中数的顺序很重要，即使数值相同，顺序不同也会导致向量不相等

import numpy as np

# 创建两个示例向量
vector1 = np.array([1, 2, 3, 4])
vector2 = np.array([1, 2, 3, 4])
vector3 = np.array([2, 1, 3, 4])

# 判断两个向量是否完全相等
print(np.array_equal(vector1, vector2))  # 输出: True
print(np.array_equal(vector1, vector3))  # 输出: False
```

```
True
False
```

练习2.1

使用numpy创建一个包含5个奇数的向量,并打印出来。

In [ ]:
```

```

练习2.2

使用numpy再创建一个和练习2.1中完全一样的向量，比较这两个向量是否相等，并打印比较结果。然后再打乱数的顺序，创建第三个向量，再比较向量是否相等，并打印比较结果。

In [ ]:
```
1
```

numpy有一些方法可以用来非常方便地创建向量

In [24]:
```
1  import numpy as np
2
3  # 创建全是0的向量
4  zero_vector = np.zeros(5)
5  print(f'zero_vector: {zero_vector}')
6
7  # 创建全是1的向量
8  one_vector = np.ones(5)
9  print(f'one_vector: {one_vector}')
10
11 # 创建全是5的向量(长度为3)
12 five_vector = np.full(3, 5)
13 print(f'five_vector: {five_vector}')
14
15 # 创建一个向量，包含10到20之间的所有整数
16 range_vector = np.arange(10, 21)
17 print(f'range_vector: {range_vector}')
18
19 # 创建一个向量，包含10到20之间的所有偶数
20 even_vector = np.arange(10, 21, 2)
21 print(f'even_vector: {even_vector}')
22
23 # 创建一个向量，包含10到20之间的所有偶数，但是逆序
24 reversed_even_vector = np.arange(20, 9, -2)
25 print(f'reversed_even_vector: {reversed_even_vector}')
26
27 # 创建包含随机数的向量，数值范围在0到1之间，长度为5
28 random_vector = np.random.random(5)
29 print(f'random_vector: {random_vector}')
```

```
zero_vector: [0. 0. 0. 0. 0.]
one_vector: [1. 1. 1. 1. 1.]
five_vector: [5 5 5]
range_vector: [10 11 12 13 14 15 16 17 18 19 20]
even_vector: [10 12 14 16 18 20]
reversed_even_vector: [20 18 16 14 12 10]
random_vector: [0.79898178 0.92198294 0.28170286 0.95456012 0.68023334]
```

练习2.3

使用numpy创建

- 一个包含4个0的向量
- 一个包含4个1的向量
- 一个包含4个2的向量
- 创建一个包含4个随机数的向量
- 创建一个包含20以内的3的倍数的整数的向量
- 创建一个包含20以内的3的倍数的整数的向量，从大到小排列

In [ ]:
```
1
```

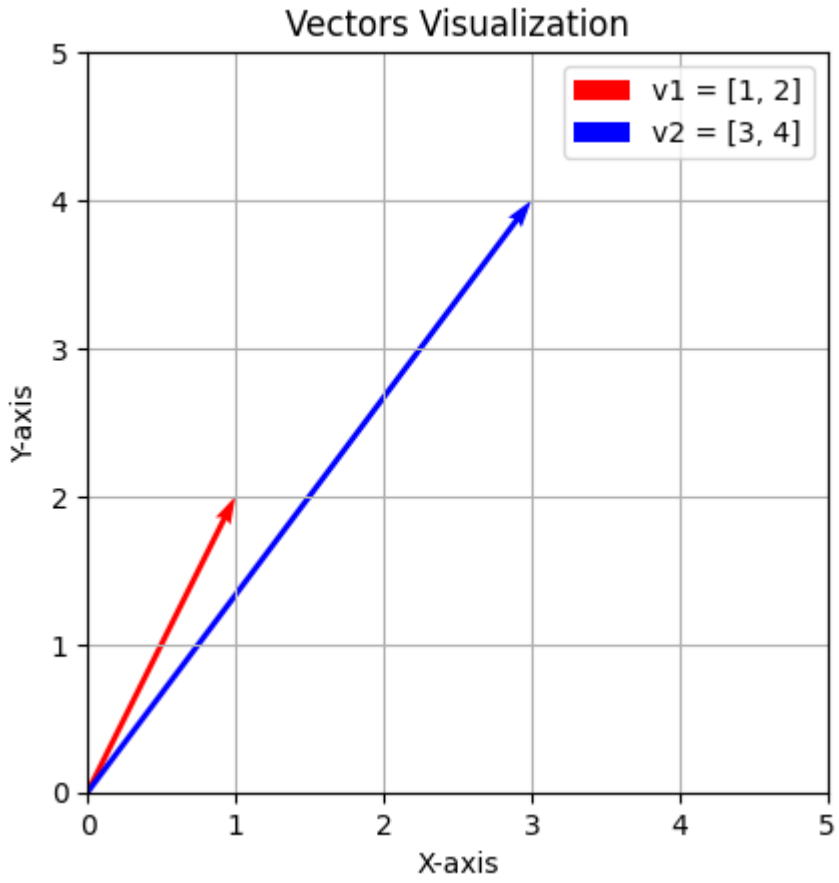
向量可视化

Python中有很多库可以用来可视化数据，比如matplotlib库。

In [ ]:
```
1 # 安装matplotlib
2 %pip install matplotlib
```

In [30]:

```python
import numpy as np
import matplotlib.pyplot as plt

# 定义向量
v1 = np.array([1, 2])
v2 = np.array([3, 4])

# 创建一个新的绘图
fig, ax = plt.subplots()

# 绘制向量 v1 和 v2
# quilt() 函数用于绘制向量(用箭头来表示)，参数含义如下:
# 0, 0 表示向量的起点在原点
# v1[0], v1[1] 表示向量的终点坐标
# angles='xy' 表示向量的起点在原点
# scale_units='xy' 表示向量的长度和方向由向量本身决定
# scale=1 表示向量的单位长度为1
# color='r' 表示向量的颜色为红色， color='b' 表示向量的颜色为蓝色
# label 表示向量的标签
ax.quiver(0, 0, v1[0], v1[1], angles='xy', scale_units='xy',
            scale=1, color='r', label='v1 = [1, 2]')
ax.quiver(0, 0, v2[0], v2[1], angles='xy', scale_units='xy',
            scale=1, color='b', label='v2 = [3, 4]')

# 设置坐标轴范围
ax.set_xlim(0, 5)
ax.set_ylim(0, 5)

ax.grid(True)   # 显示网格
ax.set_aspect('equal')   # 设置坐标轴比例相等
plt.legend()     # 显示图例
plt.xlabel('X-axis') # 设置X轴标签
plt.ylabel('Y-axis') # 设置Y轴标签
plt.title('Vectors Visualization')  # 设置标题

# 显示图形
plt.show()
```



练习2.4

使用 Matplotlib 库可视化两个二维向量。

- 创建两个 NumPy 向量 v1 和 v2，它们的元素分别是 [-2, 3] 和 [4, -1]。
- 使用 Matplotlib 库绘制这两个向量，从原点 (0, 0) 出发。
- 向量 v1 使用红色箭头表示，向量 v2 使用蓝色箭头表示。
- 设置适当的坐标轴范围，使得向量完全显示在图形中。
- 为图形添加网格、图例、坐标轴标签和标题。

In [ ]:
```
1
```

向量的数乘

向量的数乘是指一个向量乘以一个标量，结果是一个新的向量。数乘的规则是向量中的每个元素都乘以这个标量。

In [11]:
```
1 import numpy as np
2 v1 = np.array([1, 2])
3 s1 = 2
4
5 v1 * s1
```

Out[11]:
```
array([2, 4])
```

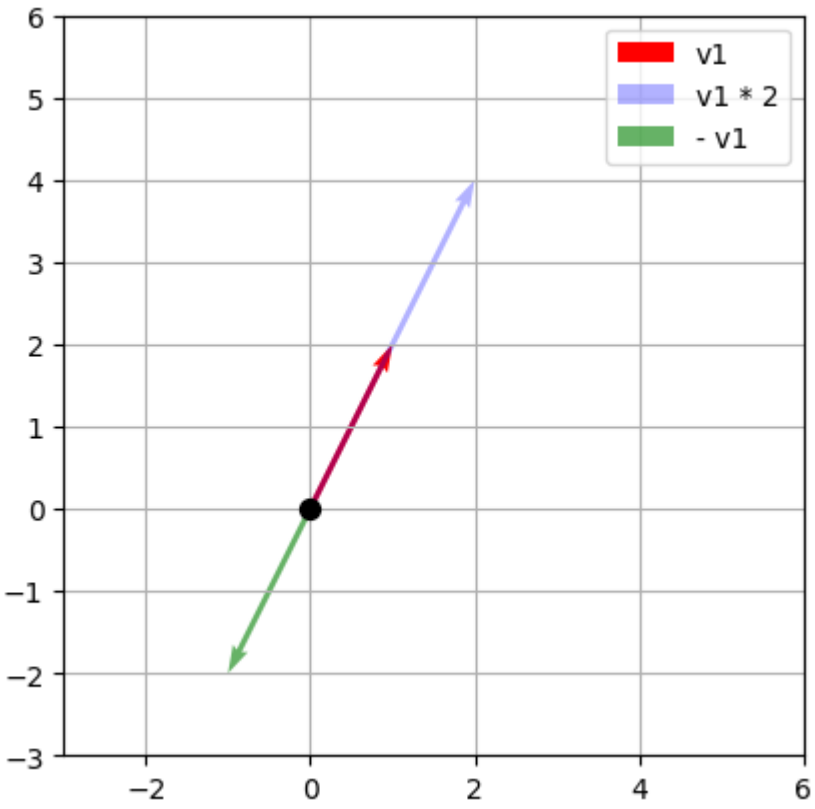
In [12]:
```
1 v1 * -1
```

Out[12]:
```
array([-1, -2])
```

向量乘以标量的可视化

In [29]:

```
import numpy as np
import matplotlib.pyplot as plt

v1 = np.array([1, 2])

v2 = v1 * 2
v3 = v1 * -1

fig, ax = plt.subplots()

plt.quiver(0, 0, v1[0], v1[1], angles='xy', scale_units='xy', scale=1,
           color='r', label='v1')

plt.quiver(0, 0,v2[0], v2[1], angles='xy', scale_units='xy', scale=1,
          color='b', label='v1 * 2', alpha = 0.3)

plt.quiver(0, 0, v3[0], v3[1], angles='xy', scale_units='xy', scale=1,
           color='g', label='- v1', alpha = 0.6)

plt.scatter(0, 0, color='k', s=50, zorder=5)

plt.xlim(-3, 6)
plt.ylim(-3, 6)
plt.grid(True)
plt.legend()

plt.gca().set_aspect('equal', adjustable='box')

plt.show()
```



练习2.5

思考下面的问题:

1. 任意一个向量乘以 0 的结果是什么?
2. 任意一个向量乘以 1 的结果是什么?
3. 任意一个向量乘以 -1 的结果是什么?
4. 一个向量v先乘以一个标量a，再乘以一个标量b，等于向量v乘以标量a*b吗？下面这些等式都成立吗？ （注意： 使用 `np.array_equal(vector1, vector2)` 来判断两个向量是否相等。）
   - $v * a * b = v * (a * b)$
   - $v * a * b = v * b * a$
5. 使用代码验证你的答案。

向量的加法和减法

相同长度的向量之间可以进行加法和减法运算

In [30]:
```
1 import numpy as np
2 v1 = np.array([1, 2])
3 v2 = np.array([3, 4])
4 v1 + v2
```

Out[30]:
```
array([4, 6])
```

In [4]:
```
1 v1 - v2
```

Out[4]:
```
array([-2, -2])
```

如果将numpy向量加上一个标量，那么向量中的每个元素都加上这个标量。numpy这个向量的特性叫做广播。

In [31]:
```
1 v1 + 2
```

Out[31]:
```
array([3, 4])
```

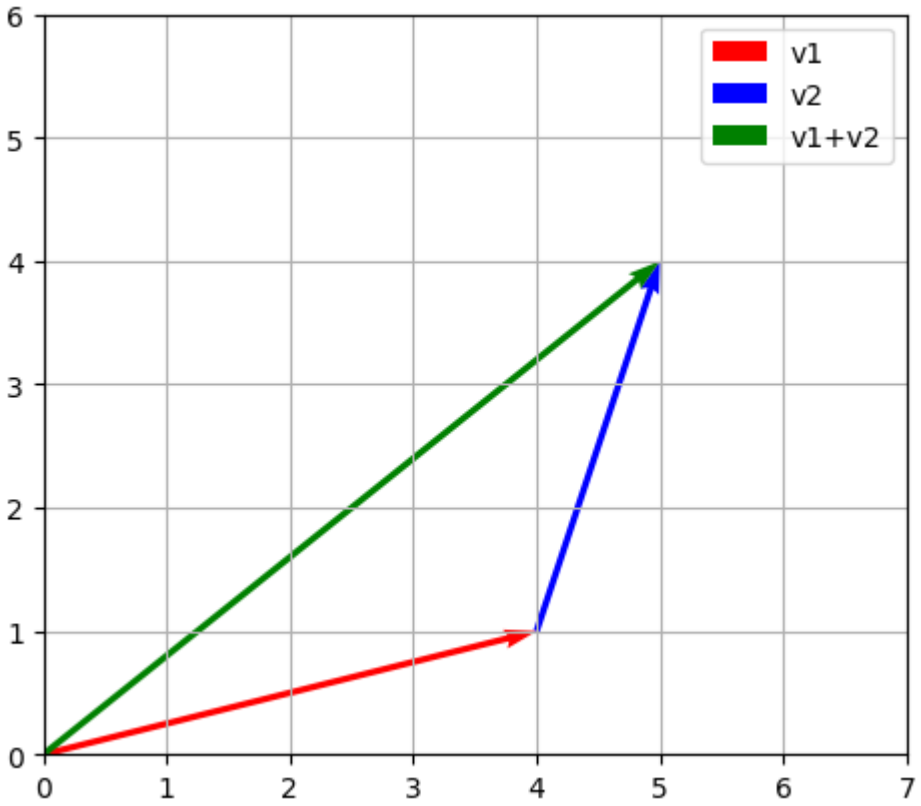
不同长度的向量之间进行加法或者减法会抛出异常

In [5]:
```
1 import numpy as np
2 v1 = np.array([1, 2])
3 v2 = np.array([3, 4, 5])
4 v1 + v2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[5], line 4
      2 v1 = np.array([1, 2])
      3 v2 = np.array([3, 4, 5])
----> 4 v1 + v2

ValueError: operands could not be broadcast together with shapes (2,) (3,)
```

向量加法的可视化

In [23]:
```
import numpy as np
import matplotlib.pyplot as plt

v1 = np.array([4, 1])
v2 = np.array([1, 3])

v3 = v1 + v2

plt.figure()
plt.quiver(0, 0, v1[0], v1[1], angles='xy', scale_units='xy', scale=1, color='r
plt.quiver(v1[0], v1[1], v2[0], v2[1], angles='xy', scale_units='xy', scale=1,
plt.quiver(0, 0, v3[0], v3[1], angles='xy', scale_units='xy', scale=1, color='g

plt.xlim(0, 7)
plt.ylim(0, 6)
plt.grid(True)
plt.legend()

plt.gca().set_aspect('equal', adjustable='box')

plt.show()
```



向量的基本运算包括：

- 向量的数乘
- 向量的加法和减法
- 以及这两种运算的结合

练习2.6

思考下面的问题：

1. 向量的加法符合交换律，$v + w = w + v$ 成立吗？（注意：使用 `np.array_equal(vector1, vector2)` 来判断两个向量是否相等。）
2. 向量的乘法符合结合分配律吗？ $a * (v + w) = a * v + a * w$ 成立吗？
3. 向量的减法都可以转化为加法运算，即 $v - w = v + (-w)$，这个等式成立吗？
4. 使用代码验证你的答案。

In [ ]:
```
1
```

向量的大小与单位向量

向量的大小(magnitude)——也称为几何长度或范数(norm)——是从向量的尾部到头部的距离，使用标准欧几里得距离公式计算：向量元素平方和的平方根。

$$\|v\| = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

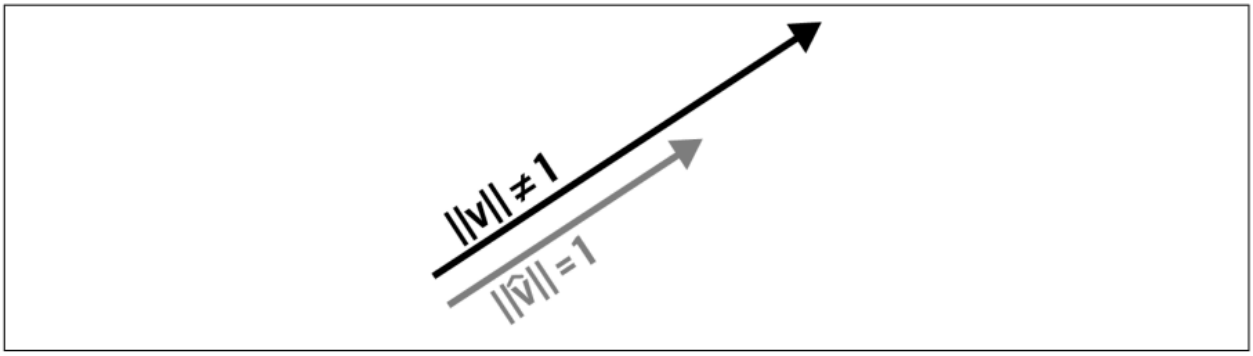
In [32]:
```
1  import numpy as np
2
3  v = np.array([1, 2, 3, 7, 8, 9])
4  v_dim = len(v)
5  v_mag = np.linalg.norm(v)
6  print(f'向量的维度: {v_dim}')
7  print(f'向量的大小: {v_mag}')
```

```
向量的维度: 6
向量的大小: 14.422205101855956
```

在某些应用中，我们需要一个几何长度为一的向量，这种向量被称为单位向量。例如在正交矩阵、旋转矩阵、特征向量和奇异向量等应用中会用到单位向量。单位向量定义为$\|v\| = 1$。计算单位向量的公式如下：

$$\hat{v} = \frac{v}{\|v\|}$$

但不是所有的向量都有单位向量。例如，零向量就没有单位向量。下图显示了一个非零非一的向量和它的单位向量。两个向量的方向相同，但长度不同。



练习2.7

使用numpy创建一个向量 `v = [3, 4]` ，计算向量的大小，并计算它的单位向量。

In [ ]:
```
1
```

向量的点积

点积（dot product），有时也称为内积(inner product),是线性代数中最重要的运算之一。它是许多操作和算法构建的基本计算构件，包括卷积、相关性、傅立叶变换、矩阵乘法、线性特征提取、信号过滤等等。

有好几种方式来表示点积，最常见的形式有：$a \cdot b$，$a^T b$，$< a, b >$。

点积的结果是一个标量，通过这一个数字可以得到两个向量之间的关系。点积的计算公式如下：

$$a \cdot b = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

例如：$[1, 2, 3] \cdot [4, 5, 6] = 1 * 4 + 2 * 5 + 3 * 6 = 32$

In [33]:
```
1 import numpy as np
2 v = np.array([1, 2, 3])
3 w = np.array([4, 5, 6])
4 np.dot(v, w)
```

Out[33]: 32

点积的一个属性是:如果使用一个标量s乘以一个向量v，然后再点积另一个向量w，等于这个标量s乘以两个向量v和w的点积。即：

$$(sv) \cdot w = s(v \cdot w)$$

In [34]:
```
import numpy as np
s = 10
v = np.array([1, 2, 3])
w = np.array([4, 5, 6])
np.dot(s * v, w)
```

Out[34]: 320

向量点积符合分配律，假设有向量a、b和c，那么：

$$a \cdot (b + c) = a \cdot b + a \cdot c$$

In [35]:
```
import numpy as np
a = np.array([0, 1, 2])
b = np.array([3, 5, 8])
c = np.array([13, 21, 34])

res1 = np.dot(a, b+c)
res2 = np.dot(a, b) + np.dot(a, c)
print(f'res1: {res1}')
print(f'res2: {res2}')
```

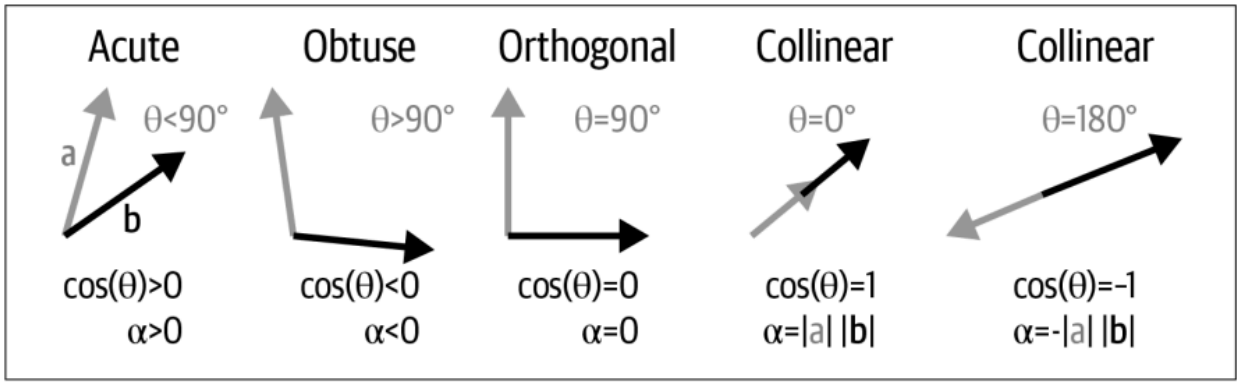
```
res1: 110
res2: 110
```

点击的几何意义是：可以反映两个向量之间的夹角的余弦值。公式如下：

$$a \cdot b = \|a\|\|b\| \cos(\theta)$$

注意向量的大小都是正数，余弦值在-1和1之间。因此：

- 如果两个向量的夹角为90度，那么点积为0。
- 如果两个向量的夹角为0度，那么点积为两个向量的大小的乘积。
- 如果两个向量的夹角为180度，那么点积为两个向量的大小的乘积的负数。
- 如果两个向量的夹角在0度到90度之间，那么点积为正数。
- 如果两个向量的夹角在90度到180度之间，那么点积为负数。



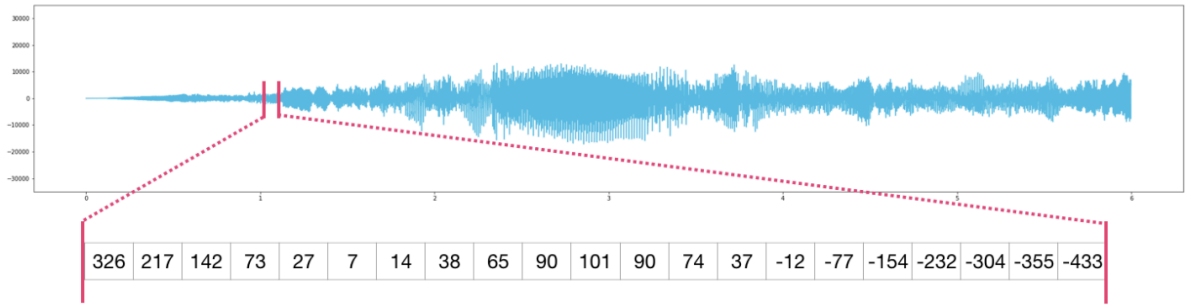
练习2.8

- 创建两个向量 `v = [1, 2, 8]` 和 `w = [4, 5, 1]`，计算这两个向量的点积

- 验证两个向量点积的符合交换律，即$a \cdot b = b \cdot a$
- 零向量与任何向量的点积都是零吗？编写代码验证一下
- 创建3个向量a、b、c，验证一下向量点积的分配律，也就是$a \cdot (b + c) = a \cdot b + a \cdot c$
- 创建两个向量 `v = [1, 2, 8]` 和 `w = [4, 5, 1]`，计算这两个向量的夹角的余弦值。
- 创建两个相互正交(垂直)的向量，计算这两个向量的点积。
- 创建两个平行的向量，计算这两个向量的点积。

In [ ]:
```
1
```

向量的应用



如何使用向量来表示音频数据：

- 采样率：音频数据是连续的，但是计算机只能处理离散的数据。因此，我们需要对音频数据进行采样，将连续的音频数据转换为离散的音频数据。采样率是指每秒钟采样的次数。例如，CD音频的采样率是44.1kHz，即每秒钟采样44100次。
- 采样值：采样值是音频数据的幅度。

In [36]:
```
1 # 加载并播放音频文件
2
3 import IPython
4 IPython.display.Audio("audio.mp3")
```

Out[36]:

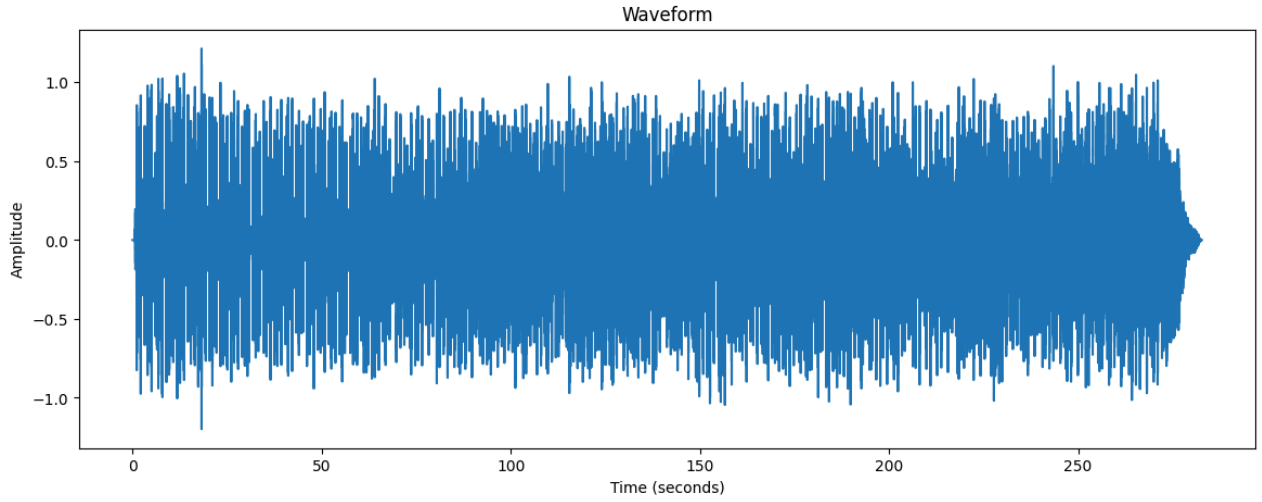
0:00 / 4:42

In [ ]:
```
1 # 下载并安装librosa（处理音频数据的Python包）
2 %pip install librosa --user
```

In [46]:
```
import librosa

# 加载音频文件
# waveform: 波形，也就是包含采样值的向量
# sample_rate: 采样率，表示每秒采样的次数
waveform, sample_rate = librosa.load('audio.mp3')
print(f'len(waveform): {len(waveform)}')
print(f'waveform[30000:30010]: {waveform[30000:30010]}')
print(f'sample_rate: {sample_rate}')
```

```
len(waveform): 6232896
waveform[30000:30010]: [0.05038331 0.05309076 0.06082571 0.06965777 0.06916472 0.0
5101117
 0.02970552 0.02174157 0.02293134 0.02447365]
sample_rate: 22050
```

In [45]:
```
# 绘制音频波形
import matplotlib.pyplot as plt
import numpy as np

plt.figure(figsize=(14, 5))
plt.plot(np.arange(len(waveform)) / sample_rate, waveform)
plt.xlabel('Time (seconds)')
plt.ylabel('Amplitude')
plt.title('Waveform')
plt.show()
```



In [ ]:
```

```

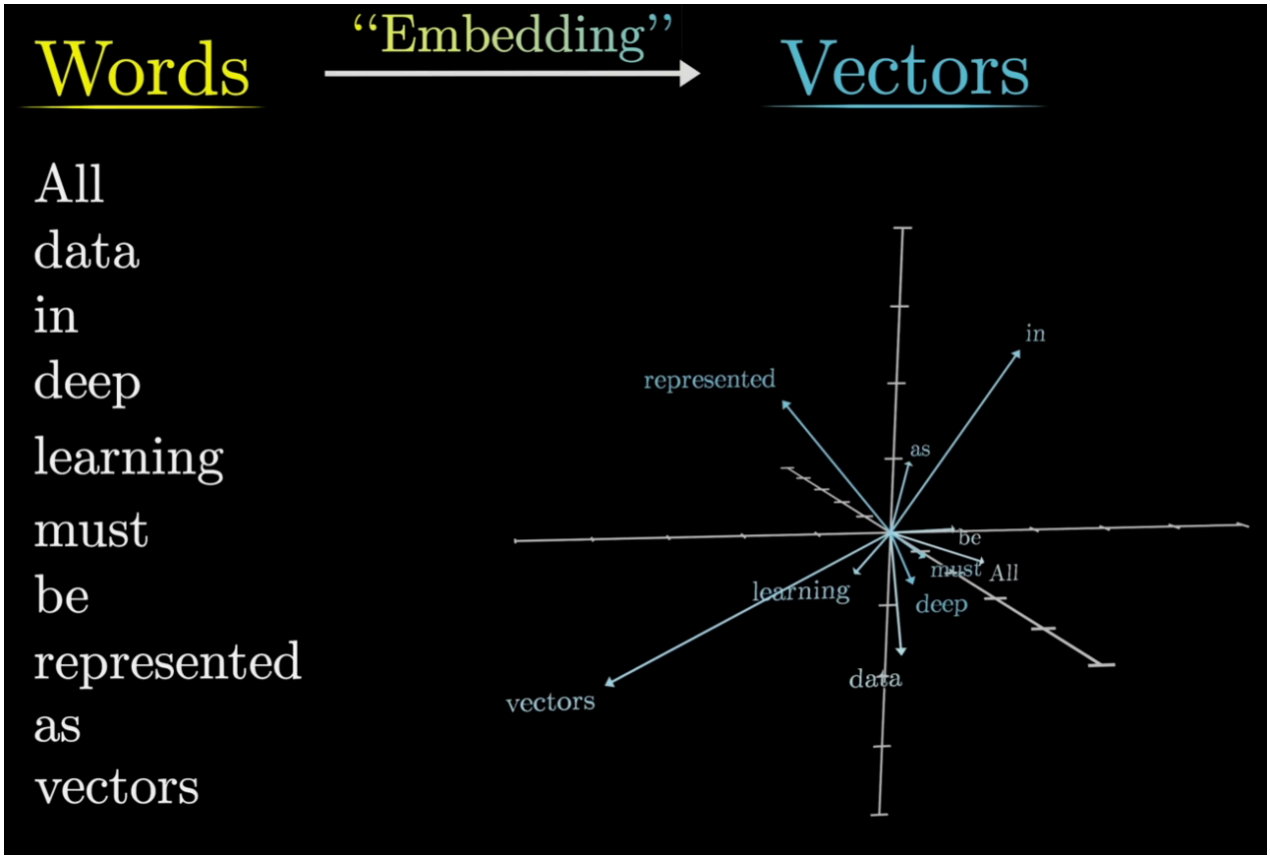
Embeddings -- 将单词表示为向量

In [ ]:
```
# install gensim
%pip install gensim
```

In [1]:
```
# 下载自然语言处理模型
import gensim.downloader
model = gensim.downloader.load('glove-wiki-gigaword-50')
```

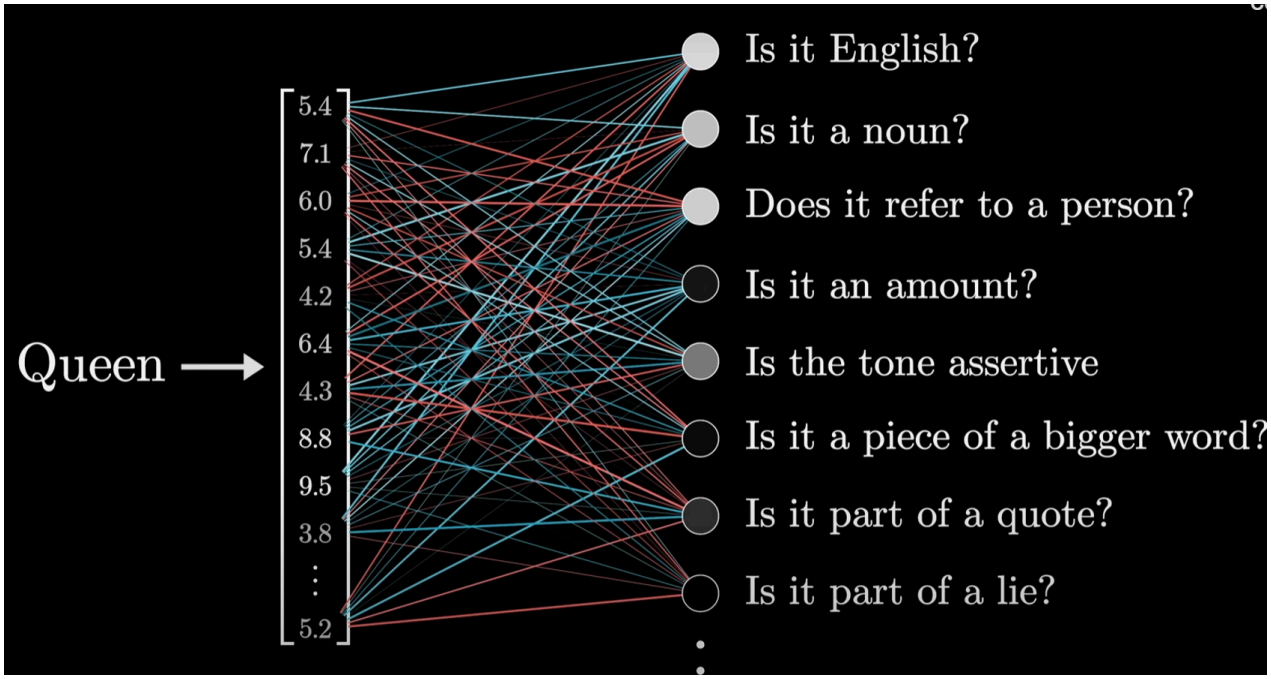
In [2]:
```
# 返回该模型包括的词汇量
len(model)
```

Out[2]:
```
400000
```



In [3]:
```
# 'glove-wiki-gigaword-50'模型每个单词的向量长度为50
# 我们没有办法直接可视化50维向量，但是可以将其投影到3维空间来显示

# GPT3 的向量长度是12,288

model['queen']
```

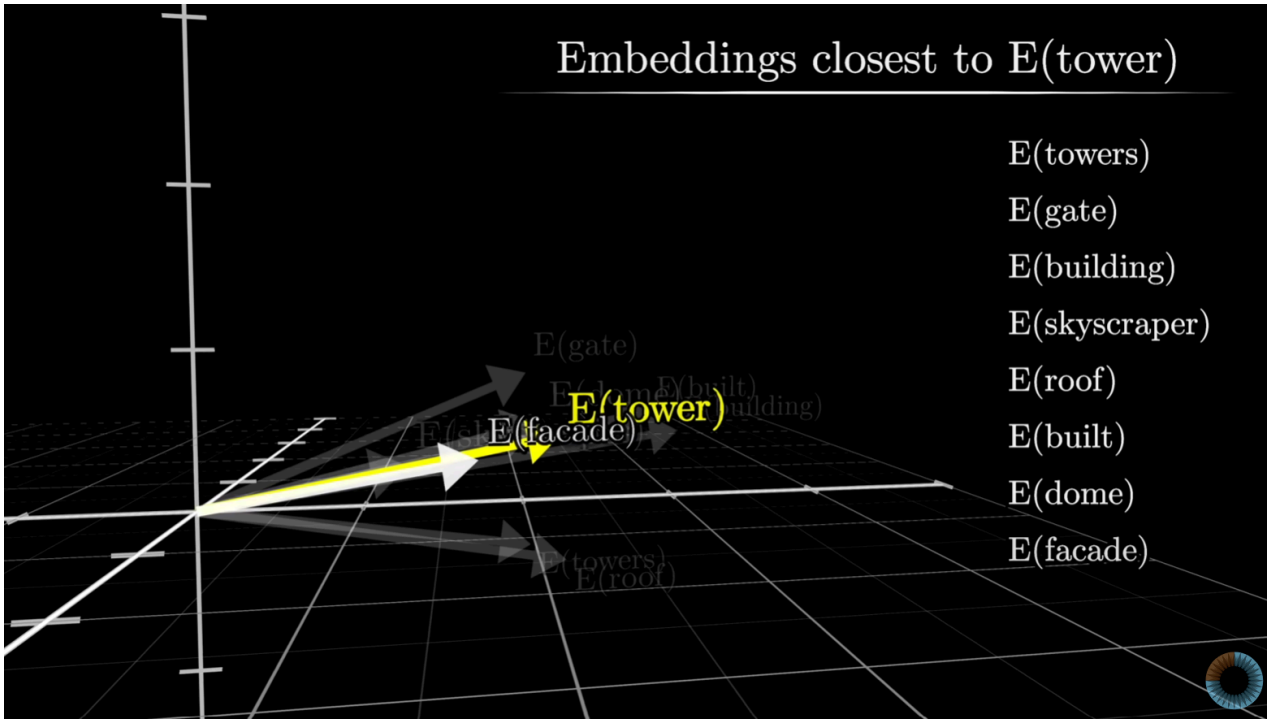
Out[3]:
```
array([ 0.37854  ,  1.8233   , -1.2648   , -0.1043   ,  0.35829  ,
        0.60029  , -0.17538  ,  0.83767  , -0.056798 , -0.75795  ,
        0.22681  ,  0.98587  ,  0.60587  , -0.31419  ,  0.28877  ,
        0.56013  , -0.77456  ,  0.071421 , -0.5741   ,  0.21342  ,
        0.57674  ,  0.3868   , -0.12574  ,  0.28012  ,  0.28135  ,
       -1.8053   , -1.0421   , -0.19255  , -0.55375  , -0.054526 ,
        1.5574   ,  0.39296  , -0.2475   ,  0.34251  ,  0.45365  ,
        0.16237  ,  0.52464  , -0.070272 , -0.83744  , -1.0326   ,
        0.45946  ,  0.25302  , -0.17837  , -0.73398  , -0.20025  ,
        0.2347   , -0.56095  , -2.2839   ,  0.0092753, -0.60284  ],
      dtype=float32)
```



- 所有的词的向量都是通过大量语料训练学习得来的
- 向量的维度表达了单词的语义信息
- 但是每个维度的语义信息都是模糊的，没有准确定义的
- 将单词转换为向量后，可以进行向量的运算，可以寻找：
  - 最接近的词
  - 最不接近的词

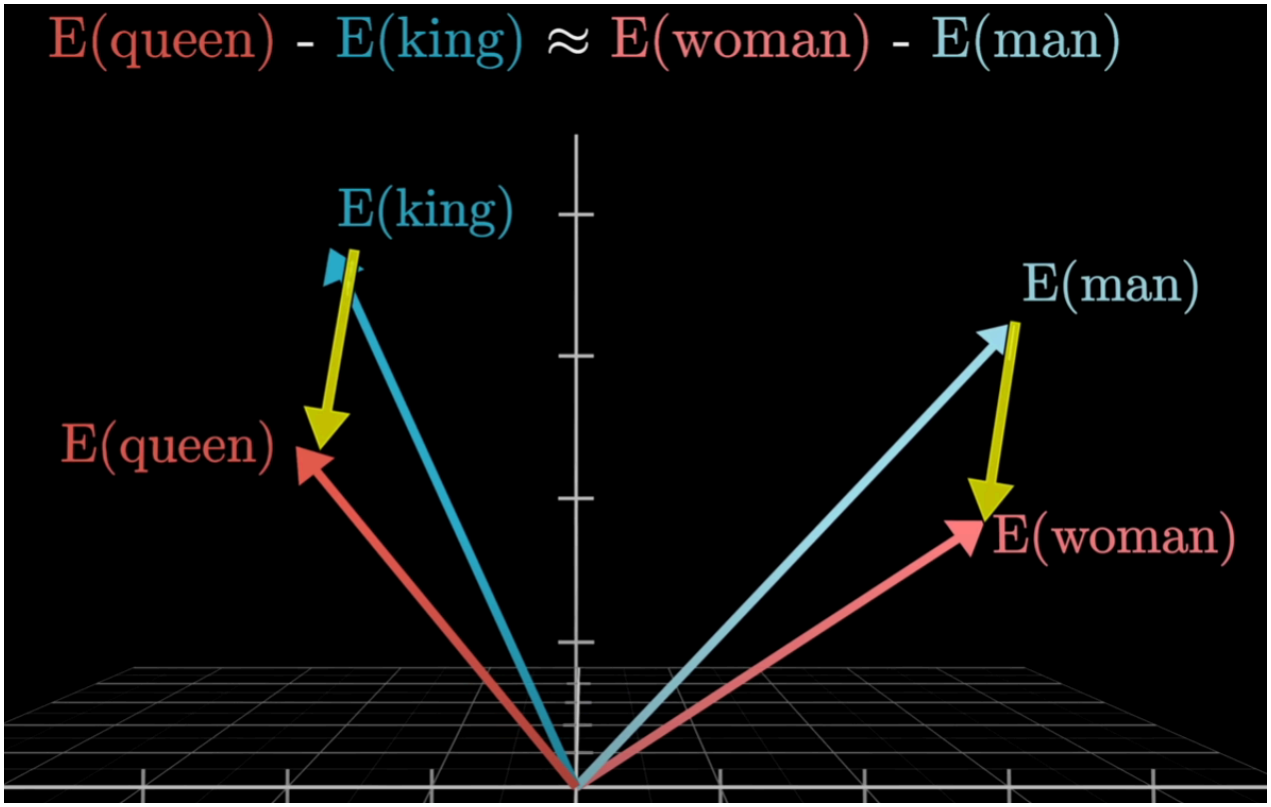
In [8]:
```
# tower 的近义词

import numpy as np
model.most_similar('tower')  # 通过点积计算相似性
```

Out[8]:
```
[('towers', 0.8470372557640076),
 ('building', 0.725898027420044),
 ('dome', 0.6875219345092773),
 ('spire', 0.6807529926300049),
 ('gate', 0.671362578868866),
 ('skyscraper', 0.6699519753456116),
 ('roof', 0.6561244130134583),
 ('walls', 0.6556639075279236),
 ('built', 0.6550073623657227),
 ('buildings', 0.6522013545036316)]
```



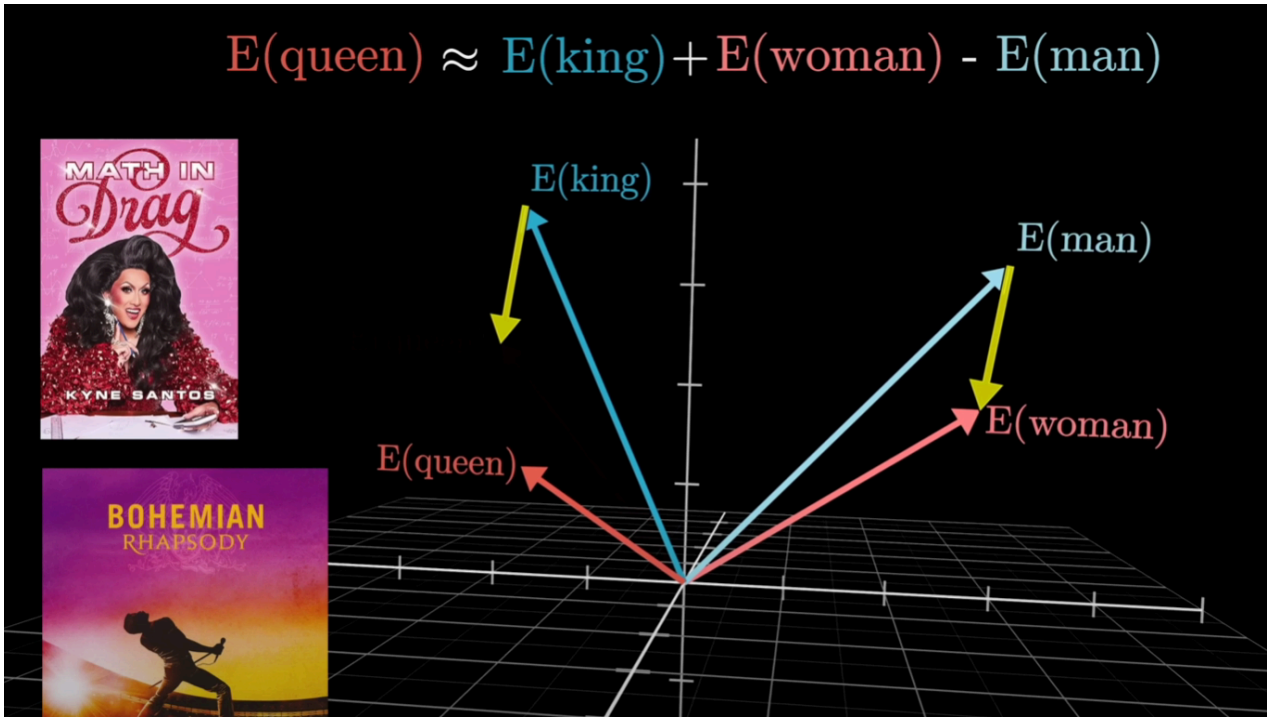
In [ ]:

```
1 # queen 的最不相似的词
2
3 model.most_similar(negative=['queen'])
```



In [9]:
```
# woman + king - man ~= queen

model.most_similar(positive=['woman', 'king'], negative=['man'])
```

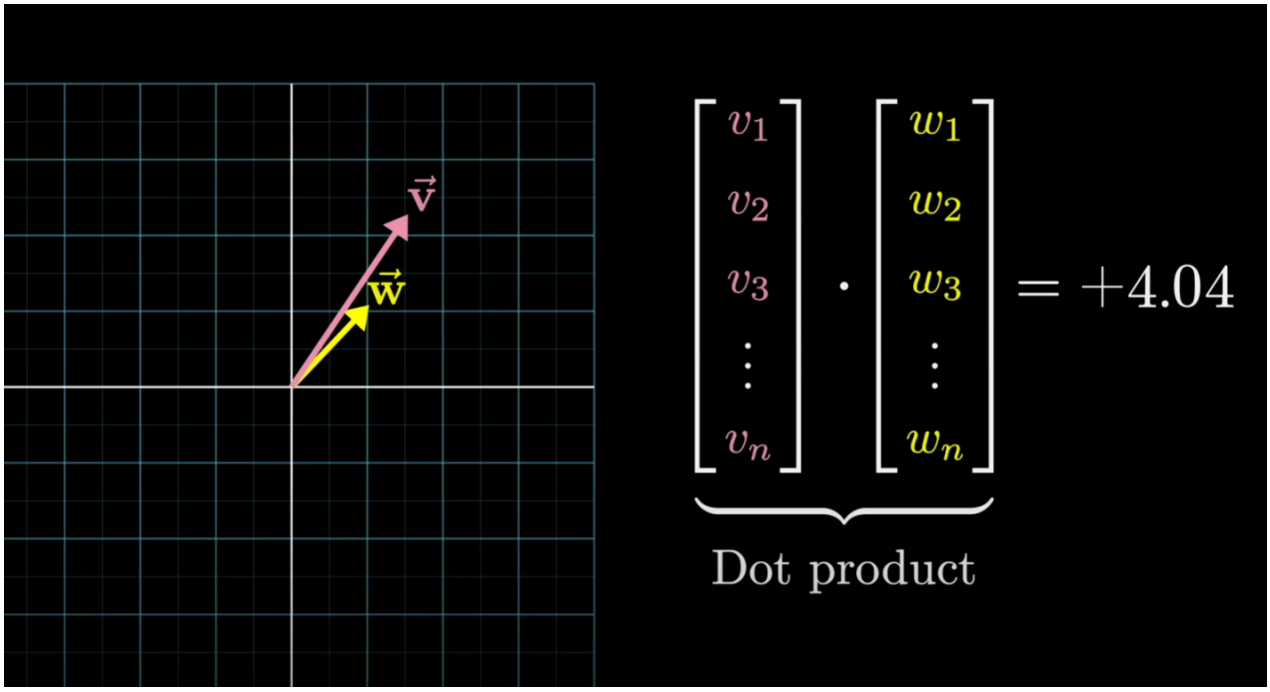
Out[9]:
```
[('queen', 0.7698540687561035),
 ('monarch', 0.6843381524085999),
 ('throne', 0.6755736470222473),
 ('daughter', 0.6594556570053101),
 ('princess', 0.6520534157752991),
 ('prince', 0.6517034769058228),
 ('elizabeth', 0.6464517712593079),
 ('mother', 0.631171703338623),
 ('emperor', 0.6106470823287964),
 ('wife', 0.6098655462265015)]
```

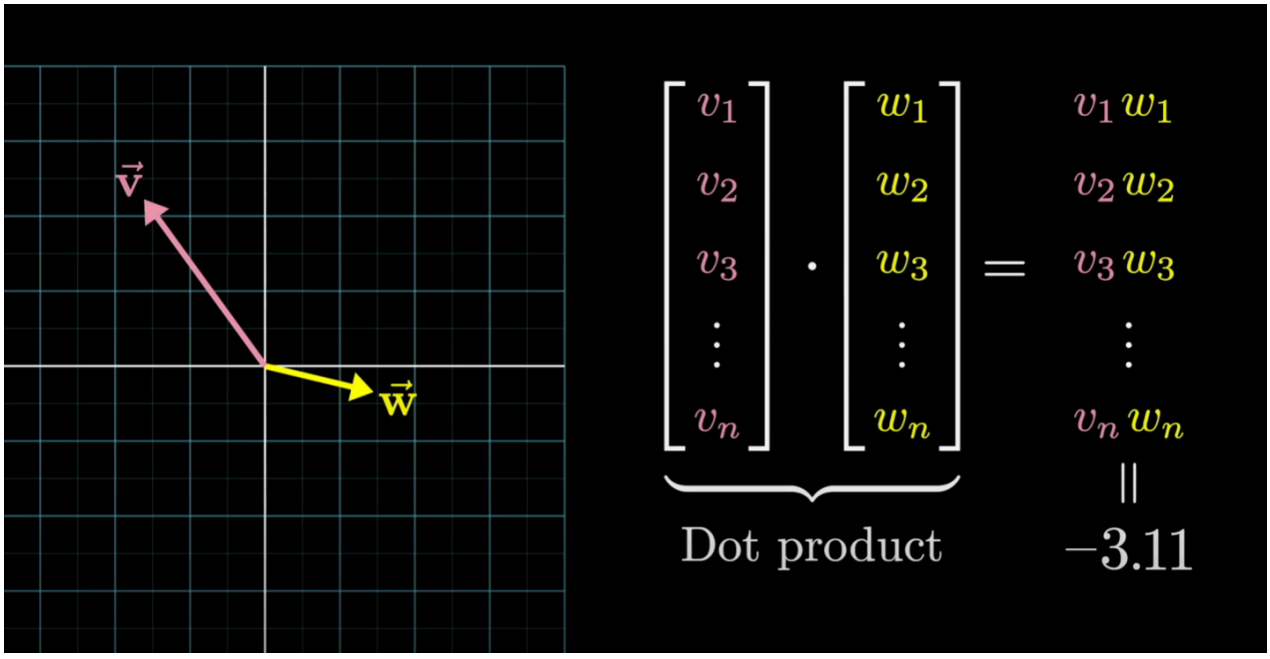


In [15]:
```
# good + happy - bad - sad ~= ?

model.most_similar(positive=['good', 'happy'], negative=['bad', 'sad'])
```

Out[15]:
```
[('enjoy', 0.4552291929721832),
 ('chance', 0.4535176753997803),
 ('ready', 0.45224252343177795),
 ('opportunity', 0.4434261918067932),
 ('excellent', 0.4415234923362732),
 ('free', 0.44127118587493896),
 ('maintain', 0.440281480550766),
 ('comfortable', 0.4352276027202606),
 ('healthy', 0.43348386883735657),
 ('better', 0.43163517117500305)]
```

如何计算两个向量之间的相似度?


$\vec{\mathbf{v}}$
$\vec{\mathbf{w}}$
$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = +4.04$
Dot product


$\vec{\mathbf{v}}$
$\vec{\mathbf{w}}$
$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} = \begin{matrix} v_1 w_1 \\ v_2 w_2 \\ v_3 w_3 \\ \vdots \\ v_n w_n \end{matrix}$
$= -3.11$
Dot product