202001555 지은미

In [13]:
```
import re, math, random
import matplotlib.pyplot as plt
from collections import defaultdict, Counter
from functools import partial, reduce
import numpy as np
```

In [14]:
```
#벡터의 덧셈

def vector_add(v,w):
    """add two vectors componentwise"""
    return [v_i + w_i for v_i, w_i in zip(v,w)]

v = [x for x in range(1,11,2)]
w = [y for y in range(11,21,2)]

vector_add(v,w)
```

Out[14]:
```
[12, 16, 20, 24, 28]
```

In [15]:
```
# numpy

np.array(v)+np.array(w)
```

Out[15]:
```
array([12, 16, 20, 24, 28])
```

In [16]:
```
# 변형

a = [x for x in range(1,10,3)]
b = [y for y in range(11,20,3)]

vector_add(a,b)
np.array(a)+np.array(b)
```

Out[16]:
```
array([12, 18, 24])
```

In [17]:
```
#시간 비교

%timeit vector_add(v,w)
%timeit np.array(v)+np.array(w)
```

```
1.33 µs ± 32.3 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
7.29 µs ± 1.46 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [18]:
```
#벡터의 뺄셈

def vector_subtract(v,w):
    """subtracts two vectors componentwise"""
    return [v_i-w_i for v_i, w_i in zip(v,w)]

vector_subtract(v,w)
```

Out[18]:
```
[-10, -10, -10, -10, -10]
```

In [19]:
```
#numpy

np.array(v)-np.array(w)
```

Out[19]:
```
array([-10, -10, -10, -10, -10])
```

In [20]:

```
# 변형

a = [x for x in range(11,20,5)]
b = [y for y in range(21,30,5)]

vector_subtract(a,b)
np.array(a)-np.array(b)
```

Out[20]:

```
array([-10, -10])
```

In [21]:

```
#벡터 리스트 덧셈

v=[x for x in range(1,11,2)]
w=[y for y in range(11,21,2)]

#Version1

def vector_sum(vectors):
    return reduce(vector_add, vectors) #reduce : 초기값을 기준으로 데이터 루프를 돌리

vectors = [v,w,v,w,v,w]
vector_sum(vectors)

#Vesion2

def vector_sum_modified(vectors):
    return [sum(value) for value in zip(*vectors)] #*:tuple 형태로 전달

vectors = [v,w,v,w,v,w]
vector_sum_modified(vectors)
```

Out[21]:

```
[36, 48, 60, 72, 84]
```

In [22]:

```
#Numpy operation

np.sum([v,w,v,w,v,w],axis=0)

#axis=0 column(열) sum operation
#axis=1 row(행) sum operation
```

Out[22]:

```
array([36, 48, 60, 72, 84])
```

In [23]:

```
#변형

a = [x for x in range(11,20,5)]
b = [y for y in range(21,30,5)]

vector=[a,b,a,a]
vector_sum(vector)
vector_sum_modified(vector)
np.sum([a,b,a,a],axis=0)
```

Out[23]:

```
array([54, 74])
```

In [24]:

```
#벡터 스칼라 곱

def scalar_multiply(c,v):
    return [c*v_i for v_i in v]

scalar=3
scalar_multiply(scalar,v)
```

Out[24]:
```
[3, 9, 15, 21, 27]
```

In [25]:
```
#Numpy Version

scalar*np.array(v)
```

Out[25]:
```
array([ 3,  9, 15, 21, 27])
```

In [26]:
```
#변형
a=[x for x in range(1,10,2)]
scalar=5
scalar_multiply(scalar,a)
scalar*np.array(a)
```

Out[26]:
```
array([ 5, 15, 25, 35, 45])
```

In [27]:
```
#벡터 리스트 평균

def vector_mean(vectors):
    """compute the vector whose i-th element is the mean of the
    i-th elements of the input vectors"""
    n=len(vectors)
    return scalar_multiply(1/n, vector_sum(vectors))

v=[1,2,3,4]
w=[-4,-3,-2,-1]

vector_mean([v,v,v,v])
```

Out[27]:
```
[1.0, 2.0, 3.0, 4.0]
```

In [28]:
```
#Numpy version
np.mean([v,v,v,v],axis=0)
```

Out[28]:
```
array([1., 2., 3., 4.])
```

In [29]:
```
#변형
a=[10,20,30,40]
b=[1,2,3,4]

vector_mean([a,a,a,a])
np.mean([a,a,a,a],axis=0)
```

Out[29]:
```
array([10., 20., 30., 40.])
```

In [30]:
```
#벡터의 내적

def dot(v,w):
    """v_1 * w_1 + ... + v_n * w_n"""
    return sum(v_i*w_i for v_i, w_i in zip(v,w))

v=[1,2,3,4]
w=[-4,-3,-2,-1]

dot(v,w)
```

Out[30]:
```
-20
```

In [31]:
```
#Numpy version
np.dot(v,w)
```

```
Out[31]: -20
```

```
In [32]: #변형
         a=[10,20,30,40]
         b=[1,2,3,4]

         dot(a,b)
         np.dot(a,b)
```

```
Out[32]: 300
```

```
In [33]: #벡터 성분 제곱 값의 합

         def sum_of_squares(v):
             """v_1 * v_1 + ... + v_n * v_n"""
             return dot(v,v)

         v=[1,2,3,4]
         sum_of_squares(v) #v*v=[1,4,9,16]
```

```
Out[33]: 30
```

```
In [34]: #변형
         a=[1,3,5,7]

         sum_of_squares(a)
```

```
Out[34]: 84
```

```
In [35]: #Magnitude (or length)
         def magnitude(v):
             return math.sqrt(sum_of_squares(v))

         magnitude(v)
```

```
Out[35]: 5.477225575051661
```

```
In [36]: #Numpy version
         np.linalg.norm(v)
```

```
Out[36]: 5.477225575051661
```

```
In [37]: #변형
         a=[1,3,5,7]
         magnitude(a)
         np.linalg.norm(a)
```

```
Out[37]: 9.16515138991168
```

```
In [38]: # 두 벡터 사이의 거리

         #original version
         def squared_distance(v,w):
             return sum_of_squares(vector_subtract(v,w))

         def distance(v,w):
             return math.sqrt(squared_distance(v,w))

         v=[1,2,3,4]
         w=[-4,-3,-2,-1]
```

```
squared_distance(v,w)
```

Out[38]:

```
100
```

In [39]:

```
distance(v,w)
```

Out[39]:

```
10.0
```

In [40]:

```
# Numpy version
np.linalg.norm(np.subtract(v,w))
```

Out[40]:

```
10.0
```

In [41]:

```
#변형
a=[11,22,33,44]
b=[-1,-2,-3,-4]
squared_distance(a,b)
distance(a,b)
np.linalg.norm(np.subtract(a,b))
```

Out[41]:

```
65.72670690061993
```

In [42]:

```
# 행렬 형태
def shape(A):
    num_rows = len(A)
    num_cols = len(A[0]) if A else 0
    return num_rows, num_cols

def get_row(A,i):
    return A[i]

def get_column(A,j):
    return [A_i[j] for A_i in A]

example_matrix = [[1,2,3,4,5],[11,12,13,14,15],[21,22,23,24,25]]

shape(example_matrix)
get_row(example_matrix,0)
get_column(example_matrix,3)
```

Out[42]:

```
[4, 14, 24]
```

In [43]:

```
# Numpy version
np.shape(example_matrix)
example_matrix=np.array(example_matrix)
example_matrix[0]
example_matrix[:,3]
```

Out[43]:

```
array([ 4, 14, 24])
```

In [44]:

```
#변형
example=[[3,6,9],[1,3,5],[4,8,12],[1,2,3]]
shape(example)
get_row(example,0)
get_column(example,2)
np.shape(example)
example=np.array(example)
example[0]
example[:,2]
```

Out[44]:

```
array([ 9,  5, 12,  3])
```

In [45]:

```
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity="all"
```

In [46]:

```
# 행렬 생성
def make_matrix(num_rows, num_cols, entry_fn): #entry_fn(i,j)= [i][j]에 해당 데이터 ㄱ
    """return a num_rows x num_cols matrix
    whose (i,j)-th entry is entry_fn(i,j)"""
    return [[entry_fn(i,j) for j in range(num_cols)]
            for i in range(num_rows)]

def is_diagonal(i,j):
    """1's on the 'diagonal', 0's everywhere else"""
    return 1 if i == j else 0

identity_matrix=make_matrix(5,5,is_diagonal)

identity_matrix
```

Out[46]:

```
[[1, 0, 0, 0, 0],
 [0, 1, 0, 0, 0],
 [0, 0, 1, 0, 0],
 [0, 0, 0, 1, 0],
 [0, 0, 0, 0, 1]]
```

In [47]:

```
# Numpy version
np.identity(5)
```

Out[47]:

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

In [48]:

```
#변형
identity_matrix=make_matrix(10,10,is_diagonal)

identity_matrix

np.identity(10)
```

Out[48]:

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 1, 0, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 1, 0, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 1, 0, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 1, 0, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0],
 [0, 0, 0, 0, 0, 0, 0, 0, 0, 1]]
```

Out[48]:

```
array([[1., 0., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0., 0., 1.]])
```

In [49]:

```
friendships=[(0,1),(0,2),(1,2),(1,3),(2,3),(3,4),(4,5),(5,6),(5,7),(6,8),(7,8),(8,9)]

friendships=[[0,1,1,0,0,0,0,0,0,0],
             [1,0,1,1,0,0,0,0,0,0],
             [1,1,0,1,0,0,0,0,0,0],
             [0,1,1,0,1,0,0,0,0,0],
             [0,0,0,1,0,1,0,0,0,0],
             [0,0,0,0,1,0,1,1,0,0],
             [0,0,0,0,0,1,0,0,1,0],
             [0,0,0,0,0,1,0,0,1,0],
             [0,0,0,0,0,0,1,1,0,1],
             [0,0,0,0,0,0,0,0,1,0]]

friendships[0][2]==1
friendships[0][8]==1

friends_of_five=[i for i, is_friend in enumerate(friendships[5]) if is_friend]
print(friends_of_five)
```

```
Out[49]: True

Out[49]: False

         [4, 6, 7]
```

In [50]:

```
#변형

friendships[0][5]==1
friendships[0][3]==1

friends_of_five=[i for i, is_friend in enumerate(friendships[7]) if is_friend]
print(friends_of_five)
```

```
Out[50]: False

Out[50]: False

         [5, 8]
```

In [51]:

```
# 행렬 덧셈
def matrix_add(A,B):
    if shape(A)!=shape(B):
        raise ArithmeticError("cannot add matrices with different shapes")

    num_rows, num_cols=shape(A)
    def entry_fn(i,j):
        return A[i][j]+B[i][j]

    return make_matrix(num_rows, num_cols, entry_fn)

A=[[1.,0.,0.],[0.,1.,2.]]
B=[[5.,4.,3.],[2.,2.,2.]]

matrix_add(A,B)
```

```
Out[51]: [[6.0, 4.0, 3.0], [2.0, 3.0, 4.0]]
```

In [52]:

```
# Numpy version
np.add(A,B)
```

```
Out[52]: array([[6., 4., 3.],
                [2., 3., 4.]])
```

In [53]:

```
#변형
A=[[1.,2.,3.],[0.,1.,2.]]
B=[[3.,4.,5.],[3.,3.,3.]]

matrix_add(A,B)
np.add(A,B)
```

```
Out[53]:  [[4.0, 6.0, 8.0], [3.0, 4.0, 5.0]]
```

```
Out[53]:  array([[4., 6., 8.],
                 [3., 4., 5.]])
```

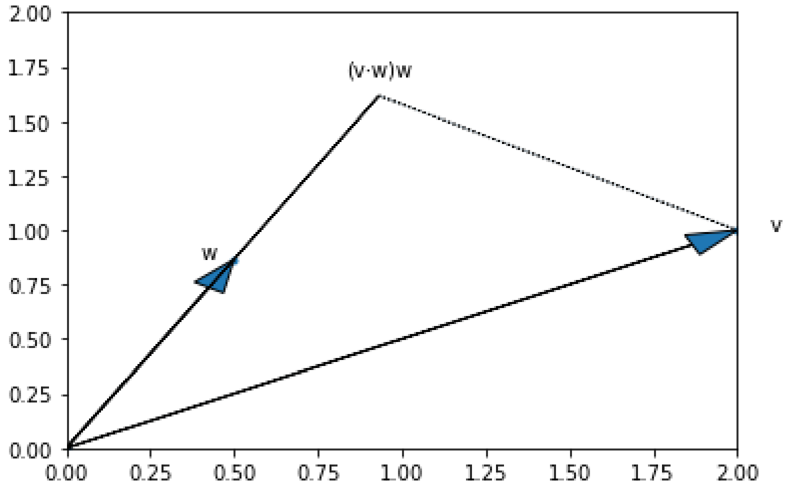
In [54]:

```
# 벡터 점곱 그래프

def make_graph_dot_product_as_vector_projection(plt):
    v=[2,1]
    w=[math.sqrt(.25), math.sqrt(.75)]
    c=dot(v,w) #벡터 내적
    vonw=scalar_multiply(c,w) #스칼라 곱
    o=[0,0]

    plt.arrow(0,0,v[0],v[1],
              width=0.002, head_width=.1, length_includes_head=True)
    plt.annotate("v",v,xytext=[v[0]+0.1,v[1]])
    plt.arrow(0,0,w[0],w[1],
              width=0.002, head_width=.1,length_includes_head=True)
    plt.annotate("w",w,xytext=[w[0]-0.1,w[1]])
    plt.arrow(0,0,vonw[0],vonw[1],length_includes_head=True)
    plt.annotate(u"(v·w)w",vonw,xytext=[vonw[0]-0.1,vonw[1]+0.1])
    plt.arrow(v[0],v[1],vonw[0]-v[0],vonw[1]-v[1],
              linestyle='dotted',length_includes_head=True)
    plt.scatter(*zip(v,w,o),marker='.')
    plt.axis([0,2,0,2])
    plt.show()

%matplotlib inline
make_graph_dot_product_as_vector_projection(plt)
```



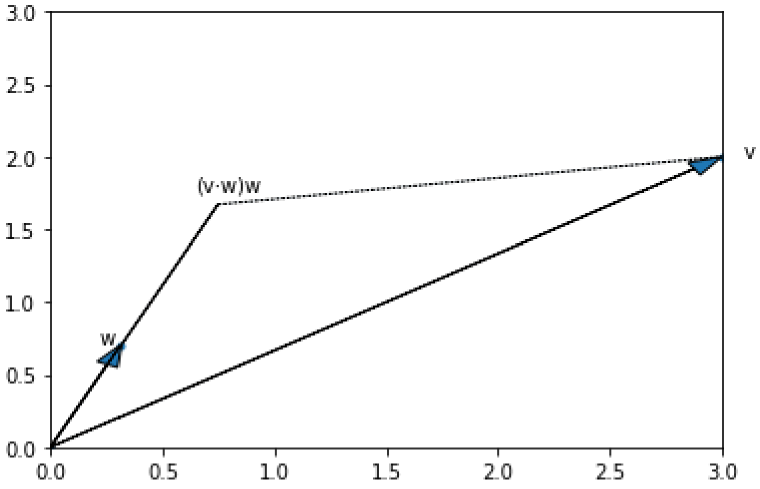
In [55]:

```
# 변형

def make_graph_dot_product_as_vector_projection(plt):
    v=[3,2]
    w=[math.sqrt(.1), math.sqrt(.5)]
    c=dot(v,w) #벡터 내적
    vonw=scalar_multiply(c,w) #스칼라 곱
    o=[0,0]

    plt.arrow(0,0,v[0],v[1],
              width=0.002, head_width=.1, length_includes_head=True)
    plt.annotate("v",v,xytext=[v[0]+0.1,v[1]])
    plt.arrow(0,0,w[0],w[1],
              width=0.002, head_width=.1,length_includes_head=True)
    plt.annotate("w",w,xytext=[w[0]-0.1,w[1]])
    plt.arrow(0,0,vonw[0],vonw[1],length_includes_head=True)
    plt.annotate(u"(v·w)w",vonw,xytext=[vonw[0]-0.1,vonw[1]+0.1])
    plt.arrow(v[0],v[1],vonw[0]-v[0],vonw[1]-v[1],
             linestyle='dotted',length_includes_head=True)
    plt.scatter(*zip(v,w,o),marker='.')
    plt.axis([0,3,0,3])
    plt.show()

%matplotlib inline
make_graph_dot_product_as_vector_projection(plt)
```



# lab5.2

In [60]:

```
# 행렬 접곱(내적)
A=[[1,2,3],[4,5,6]]
B=[[1,2],[3,4],[5,6]]

def get_column(A,j):
    return [A_i[j] for A_i in A]


def my_matrix_dot(A, B):
    array=[]
    for i in range(len(A)):
        array.append([])
        for j in range(len(B[0])):
            array[i].append(dot(A[i], get_column(B, j)))
    return array

my_matrix_dot(A, B)
```

Out[60]:

```
[[22, 28], [49, 64]]
```

In [57]:

```
# Numpy version
np.dot(A,B)
```

Out[57]:

```
array([[22, 28],
       [49, 64]])
```

In [32]:

```
# 전치 행렬

def my_matrix_transpose(M):
    T=[[0 for i in range(len(M))]for j in range(len(M[0]))]

    for i in range(len(M[0])):
        for j in range(len(M)):
            T[i][j]=M[j][i]
    return T
```

```
my_matrix_transpose(A)
my_matrix_transpose(B)
```

```
Out[32]: [[1, 4], [2, 5], [3, 6]]

Out[32]: [[1, 3, 5], [2, 4, 6]]
```

```
In [33]: # Numpy version
         np.transpose(A)
         np.transpose(B)
```

```
Out[33]: array([[1, 4],
                [2, 5],
                [3, 6]])

Out[33]: array([[1, 3, 5],
                [2, 4, 6]])
```

202001555 지은미