numpy-2

January 28, 2024

0.1 Kartezični produkti

V čisto prvi "lekciji" sem se rekel, da morata biti tabeli, ki ju želimo seštevati, množiti ali karkoli že, enakih dimenzij.

Zlagal sem se.

K tabeli z n stolpci in poljubnim številom vrstic lahko prištejemo (primnožimo, odštejemo …) enodimenzionalno tabelo z n elementi. Prištel jih bo k vsaki vrstici. Vendar nam to pri tej nalogi ne bo pomagalo. :)

Bolj zanimivo je tole:

[1]:
```
import numpy as np

f = np.array([[1], [2], [3], [4]])
g = np.array([1, 10, 100])
```

[2]:
```
f
```

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

[3]:
```
g
```

[3]:
```
array([  1,  10, 100])
```

Pomnožimo ju: `f` bo prispeval vrstice in `g` stolpce. Če se dimenzije ne ujemajo, ne bo nič hudega.

[4]:
```
f * g
```

[4]:
```
array([[  1,  10, 100],
       [  2,  20, 200],
       [  3,  30, 300],
       [  4,  40, 400]])
```

Vsak element je produkt ustreznih elementov. Ker je `f` prispeval vrstico, `g` stolpec, je

[5]:
```
(f * g)[2, 1]
```

[5]:
```
30
```

enak

[6]:
```
f[2][0] * g[1]
```

[6]:
```
30
```

(K `f[2]` smo morali dodati `[0]`, saj gre za dvodimenzionalno tabelo, mi pa bi radi prišli do števila.)

To se da nadaljevati v tretjo dimenzijo.

[7]:
```
e = np.array([[[1]], [[5]], [[8]]])
```

[8]:
```
e * f * g
```

[8]:
```
array([[[   1,   10,  100],
        [   2,   20,  200],
        [   3,   30,  300],
        [   4,   40,  400]],

       [[   5,   50,  500],
        [  10,  100, 1000],
        [  15,  150, 1500],
        [  20,  200, 2000]],

       [[   8,   80,  800],
        [  16,  160, 1600],
        [  24,  240, 2400],
        [  32,  320, 3200]]])
```

Zdaj `e` prispeva prvi indeks, `f` drugega, `g` tretjega.

[9]:
```
(e * f * g)[1, 2, 0]
```

[9]:
```
15
```

[10]:
```
e[1][0][0] * f[2][0] * g[0]
```

[10]:
```
15
```

Vrstni red množenj ni pomemben. Pomembno je, koliko dimenzij ima kdo.

[11]:
```
g * e * f
```

[11]:
```
array([[[   1,   10,  100],
        [   2,   20,  200],
        [   3,   30,  300],
        [   4,   40,  400]],
```

```
[[   5,   50,  500],
 [  10,  100, 1000],
 [  15,  150, 1500],
 [  20,  200, 2000]],

[[   8,   80,  800],
 [  16,  160, 1600],
 [  24,  240, 2400],
 [  32,  320, 3200]]])
```

0.2 Razpihovanje dimenzij

Kako si pripraviti matriko, kot je `e`? Se moramo res izgubljati v teh oglatih oklepajih? Tu nam pomaga še en trik pri indeksiranju: kot enega od indeksov lahko podamo `None`. Ta bo v tabelo, ki jo dobimo kot rezultat, dodal novo dimenzijo velikosti 1.

Tabela `e` je oblike

[12]:
```
e.shape
```

[12]:
```
(3, 1, 1)
```

Lahko bi začeli z

[13]:
```
e = np.array([1, 5, 0])

e.shape
```

[13]:
```
(3,)
```

in mu dodali manjkajoči dimenziji z dvema `None`.

[14]:
```
e[:, None, None]
```

[14]:
```
array([[[1]],

       [[5]],

       [[0]]])
```

[15]:
```
e[:, None, None].shape
```

[15]:
```
(3, 1, 1)
```

Kar smo počeli zgoraj, je torej ekvivalentno temu:

[16]:
```
f = np.array([1, 2, 3, 4])
g = np.array([1, 10, 100])
```

```
e = np.array([1, 2, 3])

e[:, None, None] * f[:, None] * g
```

[16]:
```
array([[[   1,   10,  100],
        [   2,   20,  200],
        [   3,   30,  300],
        [   4,   40,  400]],

       [[   2,   20,  200],
        [   4,   40,  400],
        [   6,   60,  600],
        [   8,   80,  800]],

       [[   3,   30,  300],
        [   6,   60,  600],
        [   9,   90,  900],
        [  12,  120, 1200]]])
```

0.3 Sploščitev matrike

Obratna operacija je `flatten`. Ta splošči tabelo v eno dimenzijo.

[17]:
```
a = np.array([[3, 6, 1], [2, 5, 7], [1, 2, 8]])

a
```

[17]:
```
array([[3, 6, 1],
       [2, 5, 7],
       [1, 2, 8]])
```

[18]:
```
a.flatten()
```

[18]:
```
array([3, 6, 1, 2, 5, 7, 1, 2, 8])
```

0.4 Naloga (drugi del)

Tale naloga ni nadležna, zahteva pa nekaj razmišljanja, nekaj znanja Pythona in spretno uporabo `numpy`-ja. Ni preprosta, ni pa ena tistih, s katerimi se zafrkavaš v nedogled in si slabe volje celo potem, ko ti uspe. Vsaj meni se ni zdela takšne.

Rešite jo, pa četudi najprej v čistem Pythonu, brez `numpy`-ja.

Pazite, v drugem delu primer iz opisa naloge uporablja druge podatke.

Spoilerji.

Trik je v tem, da bo elementov več, kot jih gre v pomnilnik. Mej med njimi pa ne. Predstavljajmo si, da vsaka koordinata, na katero naletimo pri prižiganju in ugašanju, prereže reaktor. Šestkrat -

dvakrat po `x`, dvakrat po `y`, dvakrat po `z`. Pri reševanju prvega dela je vsak element matrike ustrezal eni kocki reaktorja. Zdaj bo vsak element ustrezal celemu bloku. Bloke bo potrebno oštevilčiti, po vrsti. Slovar je vaš prijatelj, gre pa tudi z običajnimi indeksi.

Na koncu nas ne bo zanimalo število prižganih blokov temveč kock. Za to bo potrebno izračunati prostornine blokov. Tu in predvsem tu pa bo prišlo prav, kar smo se pravkar naučili.

[ ]: