VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

Dirbtinio intelekto pagrindai

Praktinė užduotis nr. 1 (Dirbtinis neuronas)

Darbą atliko:

Roland Gulbinovič

Duomenų mokslas III kursas, 2 grupė

2021

Turinys

1. Užduoties tikslas

sukurti dirbtinio neurono modelį, kuriame paduodamos įėjimų reikšmės iš duotos duomenų lentelės, nurodomos aktyvacijos funkcijos (turi būti realizuota slenkstinė ir sigmoidinė). Tinkamų svorių kombinacijų radimas. Neuronas turi paskaičiuoti išėjimo reikšmes.

x_1	x_2	**Norima reikšmė t (klasė)**
-0,2	0,5	0
0,2	-0,5	0
0,8	-0,8	1
0,8	0,8	1

1 lentelė. Duomenys

2. Neuronas su realizuota slenkstinė aktyvacijos funkcija.

$$f(a) = \begin{cases} 1, & kai\ a \geq 0 \\ 0, & kai\ a < 0 \end{cases}$$

Sukuriame slenkstinė aktyvacijos funkcija. Su argumentais $\{x_1, x_2, w_1, w_2, w_0\}$, kur x_1, x_2 – duomenys iš lentelės, w_1, w_2 – svoriai, w_0 – slenkstis.

```
def bias(x_1, x_2, w_1, w_2, w_0):
    result = x_1*w_1 + x_2*w_2 + w_0
    if result >= 0:
        return 1
    if result < 0:
        return 0
```

Pasirenku slenksti $w_0 = -0{,}2$. Sukuriu svorių intervalą nuo -1 iki 1 (-1; -0,9; -0,8...). Toliau aprašau metodus gauti visus tinkančius svorius w_1, w_2 pagal lentelės duotus duomenis.

```
w_0 = -0.2  # slenkstis

w = np.linspace(-1, 1, 21) # sukuriame svorių intervalą (-1, 1)
# po 0.1
```

```
def calc_bias0(x_1, x_2): # metodas gauti masyvą kur klasifikatoriaus reikšme
    results_w_1 = np.array([]) # Tusčias masyvas
    for i in w:
        for j in w:
            result = bias(x_1, x_2, i, j, w_0)
            if result == 0:
                results_w_1 = np.append(results_w_1, [i, j])

    n = results_w_1.size
    n = int(n/2)
    results_w_1 = results_w_1.reshape(n, 2)

    return results_w_1 # gražiname masyvą su visais svoriais, kur
klasifikatoriaus reikšme 0


def calc_bias1(x_1, x_2): # metodas gauti masyvą kur klasifikatorius reikšme
1
    results_w_1 = np.array([])
    for i in w:
        for j in w:
            result = bias(x_1, x_2, i, j, w_0) # panaudota aktyvacijos
funkcija
            if result == 1:
                results_w_1 = np.append(results_w_1, [i, j])

    n = results_w_1.size

    n = int(n/2)
    results_w_1 = results_w_1.reshape(n, 2)

    return results_w_1 # gražiname masyvą su visais svoriais kur
klasifikatorius reikšme 1



arr_1 = calc_bias0(-0.2, 0.5)
arr_2 = calc_bias0(0.2, -0.5)
arr_3 = calc_bias1(0.8, -0.8)
arr_4 = calc_bias1(0.8, 0.8)
```

Po taikomų metodų turime keturis masyvus kuriuose yra laikomi kiekvienos eilutes tinkami svoriai. Toliau lieka tiesiog palyginti masyvus ir palikti tik tuos reikšmes kurie kartojasi visuose masyvuose. Taip gauname tik tuos svorius, kurie tinka visoms keturioms eilutėms.

```
def compare(arr_1, arr_2): # Palyginimo metodas
    results_w = np.array([])
    for i in arr_1:
        for j in arr_2:
            if i[0] == j[0]:
                if i[1] == j[1]:
                    results_w = np.append(results_w, [i, j])

    n = results_w.size
```

```python
    n = int(n/2)
    results_w = results_w.reshape(n, 2)

    return results_w # gražiname pasikartojančias reikšmes

a_1 = compare(arr_1, arr_2)
a_2 = compare(arr_3, arr_4)
a = compare(a_1, a_2)
a = np.unique(a, axis = 0) # ištriname nereikalingas


a_1 = compare(arr_1, arr_2)

a_2 = compare(arr_3, arr_4)

a = compare(a_1, a_2)

a = np.unique(a, axis = 0) # istriname nereikalingas
```

Taigi gauname masyvą $\boldsymbol{a}$, kur laikomi yra visi tinkami svoriai $\{w_1, w_2\}$ su slenksčių $w_0 = -0{,}2$

```
[[ 0.3  0. ]
 [ 0.4 -0.1]
 [ 0.4  0. ]
 [ 0.4  0.1]
 [ 0.5 -0.2]
 [ 0.5 -0.1]
 [ 0.5  0. ]
 [ 0.5  0.1]
 [ 0.5  0.2]
 [ 0.6 -0.1]
 [ 0.6  0. ]
 [ 0.6  0.1]
 [ 0.6  0.2]
 [ 0.6  0.3]
 [ 0.7 -0.1]
 [ 0.7  0. ]
 [ 0.7  0.1]
 [ 0.7  0.2]
 [ 0.7  0.3]
 [ 0.7  0.4]
 [ 0.8  0. ]
 [ 0.8  0.1]
 [ 0.8  0.2]
 [ 0.8  0.3]
 [ 0.8  0.4]
 [ 0.8  0.5]
 [ 0.9  0. ]
 [ 0.9  0.1]
 [ 0.9  0.2]
 [ 0.9  0.3]
 [ 0.9  0.4]
 [ 0.9  0.5]
 [ 0.9  0.6]
 [ 1.   0.1]
 [ 1.   0.2]
 [ 1.   0.3]
 [ 1.   0.4]
 [ 1.   0.5]
 [ 1.   0.6]
 [ 1.   0.7]]
```

Taigi, galime paimti $(w_0, w_1, w_2) \Longrightarrow (-0{,}2\ , 0{,}6\ , 0{,}3)$. Galime patikrint ar tikrai gauname tokias reikšmės per kodą.

```
r_1 = bias(-0.2, 0.5, 0.6, 0.3, -0.2)
r_2 = bias(0.2, -0.5, 0.6, 0.3, -0.2)
r_3 = bias(0.8, -0.8, 0.6, 0.3, -0.2)
r_4 = bias(0.8, 0.8, 0.6, 0.3, -0.2)
print(r_1)
print(r_2)
print(r_3)
print(r_4)
0
0
1
1
```

3. Neuronas su realizuota sigmoidinė aktyvacijos funkcija.

$$f(a) = \frac{1}{1 + e^{-a}}$$

Sukuriame slenkstinė aktyvacijos funkcija. . Su argumentais $\{x_1, x_2, w_1, w_2, w_0\}$, kur x_1, x_2 – duomenis iš lentelės, w_1, w_2 – svoriai, w_0 – slenkstis. ***Laikoma kad klasė = 0, kai gauta reikšmė yra $< 0{,}5$ ir klasė = 1, kai gauta reikšmė yra $\geq 0{,}5$***

```
def sigmoid(x_1, x_2, w_1, w_2, w_0):
    result = x_1*w_1 + x_2*w_2 + w_0
    y = 1/(1 + np.exp(-result))
    if y >= 0.5:
        return 1
    if y < 0.5:
        return 0
```

Pasirenku slenksti $w_0 = -0{,}1$. Sukuriu svorių intervalą nuo -1 iki 1 (-1; -0,9; -0,8...). Toliau aprašau metodus gauti visus tinkančius svorius w_1, w_2 pagal lentelės duotus duomenys.

```
def calc_sigm0(x_1, x_2): # metodas gauti masyvą kur klasifikatoriaus reikšme
0
    results_w_1 = np.array([]) # Tusčias masyvas
    for i in w:
        for j in w:
            result = sigmoid(x_1, x_2, i, j, w_0) # panaudota aktyvacijos
funkcija
            if result == 0:
                results_w_1 = np.append(results_w_1, [i, j])

    n = results_w_1.size
    n = int(n/2)
    results_w_1 = results_w_1.reshape(n, 2)
```

```
    return results_w_1 # gražiname masyva su visais svoriais, kur gauname
klasifikatoriaus reikšme 0

def calc_sigm1(x_1, x_2): # metodas gauti masyvą reikšmių, kur
klasifikatoriaus reikšme 1
    results_w_1 = np.array([])
    for i in w:
        for j in w:
            result = sigmoid(x_1, x_2, i, j, w_0)
            if result == 1:
                results_w_1 = np.append(results_w_1, [i, j])

    n = results_w_1.size

    n = int(n/2)
    results_w_1 = results_w_1.reshape(n, 2)

    return results_w_1 # gražiname masyvą su visais svoriais kur
klasifikatorius reikšme 1
```

Po taikomų metodų turime keturis masyvus kuriuose yra laikomi kiekvienos eilutes tinkami svoriai. Toliau lieka tiesiog palyginti kurie svoriai pasitaiko kiekviename masyve. Darau taip pat kaip su slenkstinė aktyvacijos funkcija.

```
a_1 = compare(arr_1, arr_2)

a_2 = compare(arr_3, arr_4)

a = compare(a_1, a_2)
```

Taigi gauname masyvą $\boldsymbol{a}$, kur laikomi yra visi tinkami svoriai $\{w_1, w_2\}$ su slenksčių $w_0 = -0{,}1$

```
[[0.2 0. ]
 [0.3 0. ]
 [0.3 0.1]
 [0.4 0. ]
 [0.4 0.1]
 [0.4 0.2]
 [0.5 0.1]
 [0.5 0.2]
 [0.5 0.3]
 [0.6 0.1]
 [0.6 0.2]
 [0.6 0.3]
 [0.6 0.4]
 [0.7 0.1]
 [0.7 0.2]
 [0.7 0.3]
 [0.7 0.4]
 [0.8 0.2]
 [0.8 0.3]
 [0.8 0.4]
 [0.8 0.5]
 [0.9 0.2]
 [0.9 0.3]
 [0.9 0.4]
 [0.9 0.5]
 [1.  0.3]
 [1.  0.4]
 [1.  0.5]]
```

Taigi, galime paimti $\{w_0, w_1, w_2\} \Longrightarrow \{-0{,}1\ , 0{,}6\ , 0{,}4\}$. Galime patikrint ar tikrai tinkami svoriai

```
r_1 = sigmoid(-0.2, 0.5, 0.6, 0.4, -0.1)
r_2 = sigmoid(0.2, -0.5, 0.6, 0.4, -0.1)
r_3 = sigmoid(0.8, -0.8, 0.6, 0.4, -0.1)
r_4 = sigmoid(0.8, 0.8, 0.6, 0.4, -0.1)
print(r_1)
print(r_2)
print(r_3)
print(r_4)
0
0
1
1
```

4. Nelygybių sistema

$$\begin{cases} -0.2w_1 + 0.5w_2 + w_0 < 0 \\ 0.2w_1 + (-0.5)w_2 + w_0 < 0 \\ 0.8w_1 + (-0.8)w_2 + w_0 \geq 0 \\ 0.8w_1 + 0.8w_2 + w_0 \geq 0 \end{cases}$$

Nelygybių sistema pavaizduota grafinių būdų su pagalba *www.wolframalpha.com*