Sztuczna inteligencja

Propagacja wsteczna

Marcin Wojtas
Nr. albumu: s153915
III rok Niestacjonarnie
Grupa: L3

1. Cel

Celem było zaimplementowanie propagacji wstecznej w sieci neuronowej, która składała się z trzech neuronów w tym dwóch warstw, z czego pierwsza warstwa obsługująca dwa neurony była ukryta, a druga warstwa wyjściowa, w której skład wchodził jeden neuron. Neurony miały na celu uczyć poprzez wyrażenia logiczne z użyciem operatora logicznego XOR.

2. Wykorzystane pojęcia oraz wzory

Tablica prawdy dla XOR:

p	q	$p \veebar q$
0	0	0
0	1	1
1	0	1
1	1	0

Wzór na sumę ważoną w warstwach neuronów ukrytych:

$$u_i^{(1)} = \sum_{j=1}^{S} w_{ij}^{(1)} * x_j$$

Wzór na sumę ważoną w warstwach neuronów wyjściowych:

$$u_i^{(2)} = \sum_{j=1}^{q} w_{ij}^{(2)} * x$$

Wzór na błąd neuronów w warstwach wyjściowych:

$$\delta = (d_i - y_i) * f'(u)$$

Wzór na błąd neuronów w warstwach ukrytych:

$$\delta_i^{(1)} = \sum_{k=1}^{m} \delta_k^{(2)} * w_{ki}^{(2)} * f'(u)$$

Wzór dla wag w warstwach wyjściowych:

$$w_{ij}^{(2)} = w_{ij}^{(2)} + \eta * \delta_i^{(2)} * v_j^{(1)}$$

Wzór dla wag w warstwach ukrytych:

$$w_{ij}^{(2)} = w_{ij}^{(2)} + \eta * \delta_i^{(2)} * x_j$$

3. Kod źródłowy programów

```
1    import random
2    import math
```

1. Dodanie bibliotek random oraz math

```
4    learn_ratio = 0.7
5    e = 10000
6    u = [0 for j in range(3)]
7    v = [0 for j in range(3)]
8    f = [0 for j in range(3)]
9
10   x1 = [0, 0, 1, 1]
11   x2 = [0, 1, 0, 1]
12   x3 = [1, 1, 1, 1]
13   d = [0, 1, 1, 0]
14
```

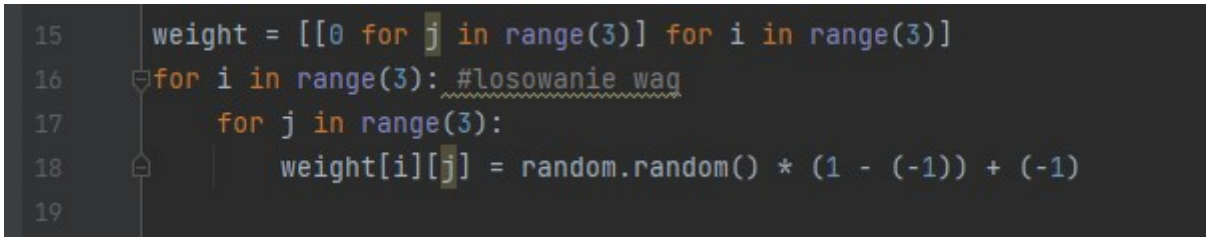
2. Utworzenie zmiennych:

2.1 **learn_ration** czyli wskaźnik uczenia,

2.2 zmiennej e reprezentującej ilość kroków,

2.3 zmiennych **u**, **v**, **f**, obejmujące po kolei sum ważonych, wyników funkcji aktywacji oraz błędów.

2.4 A także zmiennych **x1**, **x2**, **x3**, będące sygnałami wejściowymi oraz **d** który jest oczekiwanym wynikiem końcowym.



3. Dodanie tablicy dwuwymiarowej **weight** 3x3 a następnie wylosowanie wag w pętli zagnieżdżonej.

```
while e != 0:
    for i in range(4):
        u[0] = x1[i] * weight[0][0] + x2[i] * weight[0][1] + x3[i] * weight[0][2]
        u[1] = x1[i] * weight[1][0] + x2[i] * weight[1][1] + x3[i] * weight[1][2]

        v[0] = activationFunction(u[0])
        v[1] = activationFunction(u[1])

        u[2] = v[0] * weight[2][0] + v[1] * weight[2][1] + x3[i] * weight[2][2]
        v[2] = activationFunction(u[2])     #y

        f[2] = (d[i] - v[2]) * activationFunctionDerivative(v[2])

        f[0] = f[2] * weight[2][0] * activationFunctionDerivative(v[0])
        f[1] = f[2] * weight[2][1] * activationFunctionDerivative(v[1])

        weight[0][0] = weight[0][0] + learn_ratio * f[0] * x1[i]
        weight[0][1] = weight[0][1] + learn_ratio * f[0] * x2[i]
        weight[0][2] = weight[0][2] + learn_ratio * f[0] * x3[i]
        weight[1][0] = weight[1][0] + learn_ratio * f[1] * x1[i]
        weight[1][1] = weight[1][1] + learn_ratio * f[1] * x2[i]
        weight[1][2] = weight[1][2] + learn_ratio * f[1] * x3[i]
        weight[2][0] = weight[2][0] + learn_ratio * f[2] * v[0]
        weight[2][1] = weight[2][1] + learn_ratio * f[2] * v[1]
        weight[2][2] = weight[2][2] + learn_ratio * f[2] * x3[i]

        if e == 1:
            print(f"Wyjście oczekiwane: {d[i]} \t Wyjście obliczone: {v[2]}")

    e = e - 1
```

4. Pętla while wykonywana dopóki e nie osiągnie 0, w niej znajduje się petla for, jest wykonywana dla każdego z możliwych przypadków z sygnały wejściowego.
    4.1 obliczenie sumy ważonej neuronów ukrytych (u[0], u[1])
    4.2 obliczanie funkcji aktywacji neuronów ukrytych (v[0], v[1])
    4.3 obliczenie sumy ważonej oraz funkcji aktywacji dla warstw wyjściowych (u[2],v[2])
    4.4 obliczenie błędu neuronu w warstwie wyjściowej (f[2])
    4.5 obliczenie błędu neuronu w warstwie ukrytej (f[0], f[1])
    4.6 obliczenie nowych wag
    4.7 w przypadku gdy e będzie równe 1, wypisane wyścia oczekiwanego i wyjścia obliczonego
5. dekrementacja zmiennej e

```
21    def activationFunction(x): # funkcja aktywacji
22        return 1 / (1 + math.exp(-x))
23
```

6. funkcja aktywacji, zawraca wynik do określonego wzoru

```
25    def activationFunctionDerivative(x):  #pochodna
26        return x * (1 - x)
27
```

7. pochodna funkcji aktywacji

4. Przykładowe wykonania programu

Wynik dla 1000 powtórzeń:

```
Ilość kroków: 1000

Wyjście oczekiwane: 0    Wyjście obliczone: 0.09116820024258172
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9071538054936336
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9092064586018619
Wyjście oczekiwane: 0    Wyjście obliczone: 0.07881677549526339
```

Wynik dla 2500 powtórzeń:

```
Ilość kroków: 2500

Wyjście oczekiwane: 0    Wyjście obliczone: 0.031021471025123825
Wyjście oczekiwane: 1    Wyjście obliczone: 0.964266679972259
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9642392532126741
Wyjście oczekiwane: 0    Wyjście obliczone: 0.03429984229274021
```

Wynik dla 5000 powtórzeń:

```
Ilość kroków: 5000

Wyjście oczekiwane: 0    Wyjście obliczone: 0.023510365301619547
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9797900736208022
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9797615385365476
Wyjście oczekiwane: 0    Wyjście obliczone: 0.02094222284384022
```

Wynik dla 10 000 powtórzeń:

```
Ilość kroków: 10000

Wyjście oczekiwane: 0    Wyjście obliczone: 0.015068741749794858
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9835731088924495
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9835482607240419
Wyjście oczekiwane: 0    Wyjście obliczone: 0.020874927595931146
```

Wynik dla 100 000 powtórzeń:

```
Ilość kroków: 100000

Wyjście oczekiwane: 0    Wyjście obliczone: 0.004137164921924617
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9960847127135088
Wyjście oczekiwane: 1    Wyjście obliczone: 0.9952458297128823
Wyjście oczekiwane: 0    Wyjście obliczone: 0.0037042389378617027
```

5. Podsumowanie

Algorytm pozwala zaobserwować jak program dąży do uzyskania oczekiwanego wyjścia poprzez wykorzystanie operatora XOR. Duży wpływ na dążenie do oczekiwanego wyjścia ma wpływ wylosowanych wag, to zależy od nich i ilości powtórzeń jak blisko będzie wartość obliczonej do wyjściowej.

Im więcej liczby powtórzeń tym wynik jest bardziej zbliżony do oczekiwanego.

Załączniki – cały kod źródłowy (python):

```python
import random
import math

learn_ratio = 0.7
e = 10000000
u = [0 for j in range(3)]
v = [0 for j in range(3)]
f = [0 for j in range(3)]

x1 = [0, 0, 1, 1]
x2 = [0, 1, 0, 1]
x3 = [1, 1, 1, 1]
d = [0, 1, 1, 0]

weight = [[0 for j in range(3)] for i in range(3)]
for i in range(3): #losowanie wag
    for j in range(3):
        weight[i][j] = random.random() * (1 - (-1)) + (-1)


def activationFunction(x): # funkcja aktywacji
    return 1 / (1 + math.exp(-x))


def activationFunctionDerivative(x):  #pochodna
    return x * (1 - x)


print(f"Ilość kroków: {e}\n")

while e != 0:
    for i in range(4):
        u[0] = x1[i] * weight[0][0] + x2[i] * weight[0][1] + x3[i] * weight[0][2]
        u[1] = x1[i] * weight[1][0] + x2[i] * weight[1][1] + x3[i] * weight[1][2]

        v[0] = activationFunction(u[0])
        v[1] = activationFunction(u[1])

        u[2] = v[0] * weight[2][0] + v[1] * weight[2][1] + x3[i] * weight[2][2]
        v[2] = activationFunction(u[2])     #y

        f[2] = (d[i] - v[2]) * activationFunctionDerivative(v[2])

        f[0] = f[2] * weight[2][0] * activationFunctionDerivative(v[0])
        f[1] = f[2] * weight[2][1] * activationFunctionDerivative(v[1])

        weight[0][0] = weight[0][0] + learn_ratio * f[0] * x1[i]
        weight[0][1] = weight[0][1] + learn_ratio * f[0] * x2[i]
        weight[0][2] = weight[0][2] + learn_ratio * f[0] * x3[i]
        weight[1][0] = weight[1][0] + learn_ratio * f[1] * x1[i]
        weight[1][1] = weight[1][1] + learn_ratio * f[1] * x2[i]
        weight[1][2] = weight[1][2] + learn_ratio * f[1] * x3[i]
        weight[2][0] = weight[2][0] + learn_ratio * f[2] * v[0]
        weight[2][1] = weight[2][1] + learn_ratio * f[2] * v[1]
        weight[2][2] = weight[2][2] + learn_ratio * f[2] * x3[i]

        if e == 1:
            print(f"Wyjście oczekiwane: {d[i]} \t Wyjście obliczone: {v[2]}")

    e = e - 1
```

Bibliografia:

https://pl.wikipedia.org/wiki/Alternatywa_rozłączna