Ćwiczenie 1

Rozwiązanie zagadnienia klasyfikacji za pomocą pojedynczego neuronu uczonego metodą bezgradientową

Spis treści

Cel ćwiczenia

Celem zadania było zaprojektowanie modelu neuronu (bazującego na modelu McCullocha-Pittsa) z dwoma wejściami, który potrafi klasyfikować punkty należące do dwóch liniowo separowalnych zbiorów. Po zakończeniu procesu uczenia sprawdzono, z jaką skutecznością neuron poprawnie klasyfikuje dostarczone punkty.

Przebieg ćwiczenia

Przygotowanie zbiorów

Przygotowano dwa zbiory punktów liniowo separowalnych, oparte na prostej $y = -5x + 5$, o licznościach sample_size = {20, 50, 100, 500}. Współrzędne punktów były ograniczone do zakresu $-15 \leq x \leq 15$ oraz $-15 \leq y \leq 15$. Punkty z tych zbiorów zostały podzielone na zbór uczący (train) i zbór weryfikacyjny (test). Każdy punkt otrzymał etykietę z zakresu {0, 1}, gdzie 1 oznacza punkt znajdujący się powyżej prostej, a 0 oznacza punkt znajdujący się poniżej prostej.

Zaprojektowanie perceptronu

Zaprojektowano perceptron z dwoma wejściami i funkcją aktywacji Heaviside'a, która przypisuje wartość 1 dla parametru o wartości większej niż 0, a wartość 0 dla pozostałych przypadków:

$$f(x) = \begin{cases} 1 \; dla \; x > 0 \\ 0 \; dla \; x \leq 0 \end{cases}$$

Wagi perceptronu zostały zainicjowane losowymi wartościami.

Proces nauczania

Proces nauczania perceptronu przeprowadzono metodą bezgradientową, wykorzystując dane ze zbioru train. Procedura nauki polegała na obliczaniu wyjściowych wartości neuronu dla wszystkich danych uczących, a następnie korekcji wag perceptronu dla każdego punktu wejściowego, w którym wystąpił błąd. Proces kontynuowano do momentu, gdy dla całej epoki nie wystąpiła potrzeba modyfikacji żadnej wagi.

Proces weryfikacji

Po zakończeniu procesu nauczania przeprowadzono weryfikację, korzystając ze zbioru test jako danych wejściowych. Sprawdzono, jaki procent punktów ze zbioru testowego został poprawnie sklasyfikowany przez perceptron.

Uwagi

W celu zapewnienia powtarzalności wyników, na początku programu ustawiono wartość „seed" dla generatora liczb pseudolosowych, co gwarantowało identyczne wyniki przy każdym uruchomieniu programu.

Program został napisany w języku Python, z wykorzystaniem biblioteki numpy.

Kod źródłowy programu

```python
import numpy as np
import random

class Perceptron:
    def __init__(self, num_inputs):
        self.b = np.random.rand(1)[0]
        self.weights = np.random.rand(num_inputs)

    def activate(self, value):
        return 1 if value >= 0 else 0

    def linear_combination(self, input_data):
        return np.sum(self.weights * input_data) + self.b

    def predict(self, input_data):
        return self.activate(self.linear_combination(input_data))

    def learn(self, training_data, expected_output):
        if len(training_data) != len(expected_output):
            return
```

```
21.
22.         total_samples = len(training_data)
23.
24.         correctly_classified = None
25.         while correctly_classified != total_samples:
26.             correctly_classified = 0
27.             predictions = [self.predict(data_point) for data_point in training_data]
28.
29.             for idx in range(total_samples):
30.                 error = expected_output[idx] - predictions[idx]
31.
32.                 if error == 0:
33.                     correctly_classified += 1
34.                     continue
35.
36.                 self.weights += np.array(training_data[idx]) * error
37.                 self.b += error
38.
39.     def verify(self, test_data, expected_output):
40.         correct_predictions = 0
41.
42.         for idx in range(len(test_data)):
43.             predicted = self.predict(test_data[idx])
44.             correct_predictions += 1 * (predicted == expected_output[idx])
45.
46.         return correct_predictions
47.
48.     def __str__(self):
49.         return 'Perceptron parameters:\n\n' + \
50.                '\tweights: ' + str(self.weights) + '\n' \
51.                + '\tbias = ' + str(self.b) + '\n'
52.
53. def create_points(point_count, slope, intercept, range_limits):
54.     data_points = []
55.     labels = []
56.
57.     created_points = 0
58.     while created_points < point_count:
59.         x_coord = random.randint(*range_limits)
60.         y_coord = random.randint(*range_limits)
61.         y_threshold = slope * x_coord + intercept
62.
63.         label = 1 * (y_coord > y_threshold)
64.
65.         if (created_points < point_count / 2) != (label == 0):
66.             continue
67.
68.         data_points.append((x_coord, y_coord))
69.         labels.append(label)
70.         created_points += 1
71.
72.     return data_points, labels
73.
74. if __name__ == '__main__':
75.     random.seed(1928342)
76.
77.     # Parameters
78.     sample_size = 500
79.     coordinate_range = (-15, 15)
80.     line_slope, line_intercept = 5, 5
81.
```

```
82.     # Training and testing sets
83.     train_points, train_labels = create_points(sample_size, line_slope, line_intercept,
coordinate_range)
84.     test_points, test_labels = create_points(sample_size, line_slope, line_intercept,
coordinate_range)
85.
86.     print('Generated', str(sample_size), 'training points:')
87.     for item in zip(train_points, train_labels):
88.         print(item)
89.
90.     node = Perceptron(2)
91.     node.learn(train_points, train_labels)
92.
93.     print('\nLearning is finished.\n')
94.     print(node)
95.
96.     verification_result = node.verify(test_points, test_labels)
97.     print('Correctly classified {} out of {} test points ({:.2f}% accuracy)'
98.           .format(verification_result, len(test_points), verification_result /
len(test_points) * 100))
99.
```

Wyniki

Przedstawione wyniki osiągnięto, ustawiają wartość „seed” na 1928342 oraz zmieniając liczbę sample_size losowo wygenerowanych punktów.

Wynik dla parametru sample_size = 20:

Generated 20 training points:
((-1, -7), 0)
((1, -8), 0)
…
((-7, 14), 1)

Learning is finished.

Perceptron parameters:

weights: [-134.11445812 36.58202985]
bias = 1.714389405775382

Correctly classified **19** out of **20** test points (**95.00%** accuracy)

Wynik dla parametru sample_size = 50:

Generated 50 training points:
((-1, -7), 0)
((2, 5), 0)
…
((-9, -1), 1)

Learning is finished.

Perceptron parameters:

weights: [-200.55745117 47.95946872]
bias = -35.368163307793154

Correctly classified **49** out of **50** test points (**98.00%** accuracy)

Wynik dla parametru sample_size = 100:

Generated 100 training points:
((-7, -3), 1)
((-7, -6), 1)
…
((-12, -3), 1)

Learning is finished.

Perceptron parameters:

weights: [-650.28824945 145.58796375]
bias = -157.34884450549004

Correctly classified **99** out of **100** test points (**99.00%** accuracy)

Wynik dla parametru sample_size = 500:

Generated 500 training points:
((-15, -7), 1)
((-13, -3), 1)
…
((-1, 2), 1)

Learning is finished.

Perceptron parameters:

weights: [-1539.83387471 308.78052021]
bias = -1553.2314905567562

Correctly classified **500** out of **500** test points (**100.00%** accuracy)

Wnioski

- Przy mniejszych wartościach parametru sample_size (20, 50, 100) liczba danych uczących była niewystarczająca, aby osiągnąć 100% dokładności.
- Wynik procesu uczenia zależy od losowych wartości początkowych wag neuronu oraz rozmieszczenia punktów uczących.
- Gdy parametr sample_size wynosi 500, zagęszczenie punktów w określonym zakresie współrzędnych wzrasta, co zwiększa szansę algorytmu uczącego na bardziej precyzyjne dopasowanie wag perceptronu.

Autor

Łukasz Kałużny, numer indeksu: 240695, grupa 7-IO