Neuronale Netze - Übung 5

Tobias Hahn
3073375

November 18, 2016

1 Perzeptionsalgorithmus

1.1 Ziffernerkenner

Der Ziffernerkenner wurde in Python implementiert. Die Fehlerraten sind ziemlich hoch, was darauf hinweist dass der Lernalgorithmus für diese Aufgabe ungeeignet ist. Das liegt daran dass der Algorithmus nur die Richtung eines Zahlenclusters vom Nullpunkt angeben kann, da der Vektor nach jedem Schritt normalisiert wird. Da die Ziffern sich in der Richtung jedoch teilweise start überlappen (so ist die Form der 3 eine Unterform der 8 und weist damit in die selbe Richtung) können sie nicht gut auseinandergehalten werden. Hier wäre Lineare Regression besser geeignet, da hier nicht nur die Richtung, sondern der Zentrumspunkt der Zahlen unterschieden wird, also auch der Abstand.

Die Tabellen in der Abgabe werden vom Programm auch so ausgegeben, wobei die erste Zeile die Headerzeile ist und angibt was sich darunter befindet (die Nummer des Vektors, danach die Ziffern wie in der vorgegebenen Klassifizierung). Das Array vor den Tabellen gibt an welche Ziffer der Vektor am meisten erkannt hat, welche Ziffer er also klassifizert wenn er angewandt wird (also sein Produkt am größten ist).

Im folgenden zuerst der Quellcode für die beiden Klassen, danach die Ausgabe in der Kommandozeile.

../konkurrenz.py

```python
import numpy as np
from texttable import Texttable

def clustering(digits, k):
    dim_num = len(digits['data'][0])
    digits_number = len(digits)
    weight_vecs = np.random.rand(k, dim_num)
    np.apply_along_axis(np.linalg.norm, 1, weight_vecs)

    for i in range(0,100000):
        current_instance = digits['data'][np.random.randint(0,digits_number)]
        largest_index = -1
        largest_value = float("-inf")

        for j in range(0,k):
            current_value = np.dot(current_instance, weight_vecs[j])
            if (current_value > largest_value):
                largest_value = current_value
                largest_index = j

        temp_vector = weight_vecs[largest_index] + current_instance
        weight_vecs[largest_index] = temp_vector / np.linalg.norm(temp_vector)

    return assign_numbers(digits, weight_vecs)

def assign_numbers(digits, weight_vecs):
    count = np.zeros((len(weight_vecs), 10))

    for digit in digits:
        largest_index = -1
        largest_value = float("-inf")

        for j in range(0,len(weight_vecs)):
            current_value = np.dot(digit['data'], weight_vecs[j])
            if (current_value > largest_value):
                largest_value = current_value
                largest_index = j

        count[largest_index][digit['value']] += 1

    named_vecs = np.zeros(len(weight_vecs), dtype=[('vector', 'f', len(weight_vecs[0])), ('
        digit', 'i')])

    for i,v in enumerate(named_vecs):
        named_vecs['vector'][i] = weight_vecs[i]
        named_vecs['digit'][i] = np.argmax(count[i])

    print(named_vecs['digit'])

    return named_vecs

def predict_number(digit, named_vecs):
    largest_index = -1
    largest_value = float("-inf")

    for j in range(0,len(named_vecs)):
        current_value = np.dot(digit, named_vecs['vector'][j])
        if (current_value > largest_value):
            largest_value = current_value
            largest_index = j

    return named_vecs['digit'][j]

def calc_error(digits, named_vecs):
    error = 0

    for d in digits:
        if (d['value'] != predict_number(d['data'], named_vecs)):
            error += 1

    return error


def print_results(digits, weight_vecs):
    count = np.zeros((len(weight_vecs), 10))

    for digit in digits:
        largest_index = -1
        largest_value = float("-inf")

        for j in range(0,len(weight_vecs)):
            current_value = np.dot(digit['data'], weight_vecs[j])
            if (current_value > largest_value):
                largest_value = current_value
                largest_index = j

        count[largest_index][digit['value']] += 1

    t = Texttable()
    t.add_row(["Vektor", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"])
    for i,v in enumerate(count):
        t.add_row(np.insert(v, 0, i))
    print t.draw()
```

../train.py

```python
import numpy as np
import konkurrenz as ko

def extractDigits(filename, expected_num):
    data_count = 0
    digit_count = 0
    data_points_per_digit = 192
    data_points_per_line = 12

    digits = np.zeros(expected_num, dtype=[('data', 'f', data_points_per_digit), ('value', 'i')])

    with open(filename) as f:
        lines = f.readlines()

    for i,line in enumerate(lines):
        digits_line = line.split()
        if (len(digits_line) == data_points_per_line):
            for num in digits_line:
                digits['data'][digit_count][data_count] = float(num)
                data_count += 1
        elif (len(digits_line) == 10):
            for i,num in enumerate(digits_line):
                if (num == "1.0"):
                    digits['value'][digit_count] = i
                    break
        else:
            if (data_count == data_points_per_digit and digit_count < expected_num):
                digit_count += 1
                data_count = 0
            else:
                print("Exited␣because␣of␣wrong␣data")
                raise SystemExit

    if (digit_count == expected_num):
        return digits
    else:
        print("Exited␣because␣of␣few␣digits")
        raise SystemExit

if __name__ == "__main__":
    training_name = "./data/digits.trn"
    training_number = 1000
    training_digits = extractDigits(training_name, training_number)

    test_name = "./data/digits.tst"
    test_number = 200
    test_digits = extractDigits(test_name, test_number)

    named_vectors = ko.clustering(training_digits, 10)
    ko.print_results(training_digits, named_vectors['vector'])
    print("On␣the␣training␣set,␣the␣algorithm␣made␣{0}␣mistakes␣for␣{1}␣digits.".format(ko.
        calc_error(training_digits, named_vectors), len(training_digits)))
    print("On␣the␣test␣set,␣the␣algorithm␣made␣{0}␣mistakes␣for␣{1}␣digits.".format(ko.
        calc_error(test_digits, named_vectors), len(test_digits)))

    named_vectors = ko.clustering(training_digits, 12)
    ko.print_results(test_digits, named_vectors['vector'])
    print("On␣the␣training␣set,␣the␣algorithm␣made␣{0}␣mistakes␣for␣{1}␣digits.".format(ko.
        calc_error(training_digits, named_vectors), len(training_digits)))
    print("On␣the␣test␣set,␣the␣algorithm␣made␣{0}␣mistakes␣for␣{1}␣digits.".format(ko.
        calc_error(test_digits, named_vectors), len(test_digits)))
```

Beispielausgabe

```
[7 1 7 4 7 1 2 3 8 0]
+--------+-----+----+----+----+----+----+----+----+----+----+
| Vektor | 0   | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 0      | 0   | 0  | 5  | 2  | 0  | 0  | 0  | 6  | 0  | 2  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 1      | 1   | 26 | 13 | 1  | 7  | 0  | 1  | 3  | 1  | 0  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 2      | 0   | 6  | 6  | 5  | 8  | 7  | 0  | 50 | 2  | 37 |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 3      | 15  | 2  | 1  | 0  | 60 | 5  | 39 | 1  | 0  | 2  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 4      | 0   | 4  | 4  | 2  | 17 | 7  | 0  | 33 | 1  | 7  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 5      | 1   | 67 | 1  | 0  | 2  | 0  | 2  | 0  | 2  | 2  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 6      | 0   | 1  | 29 | 0  | 1  | 1  | 0  | 4  | 0  | 0  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 7      | 0   | 0  | 13 | 77 | 0  | 3  | 1  | 0  | 3  | 8  |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 8      | 30  | 3  | 9  | 26 | 4  | 37 | 26 | 1  | 70 | 64 |
+--------+-----+----+----+----+----+----+----+----+----+----+
| 9      | 106 | 0  | 1  | 0  | 1  | 6  | 6  | 0  | 0  | 3  |
+--------+-----+----+----+----+----+----+----+----+----+----+
On the training set, the algorithm made 847 mistakes for 1000 digits.
On the test set, the algorithm made 167 mistakes for 200 digits.
```

```
[1 8 1 0 7 5 6 9 7 4 5 5]
+--------+----+----+---+---+----+---+---+----+---+---+
| Vektor | 0  | 1  | 2 | 3 | 4  | 5 | 6 | 7  | 8 | 9 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 0      | 0  | 15 | 5 | 8 | 0  | 0 | 0 | 0  | 3 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 1      | 0  | 0  | 0 | 0 | 0  | 1 | 0 | 0  | 5 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 2      | 0  | 0  | 0 | 0 | 0  | 0 | 0 | 0  | 0 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 3      | 32 | 0  | 2 | 4 | 1  | 1 | 4 | 1  | 1 | 9 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 4      | 0  | 1  | 2 | 0 | 0  | 0 | 0 | 5  | 0 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 5      | 0  | 1  | 0 | 0 | 0  | 2 | 0 | 0  | 0 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 6      | 1  | 0  | 6 | 6 | 3  | 3 | 8 | 0  | 0 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 7      | 0  | 3  | 1 | 2 | 6  | 3 | 0 | 0  | 1 | 8 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 8      | 0  | 0  | 6 | 3 | 0  | 0 | 0 | 14 | 0 | 1 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 9      | 0  | 0  | 0 | 0 | 12 | 0 | 1 | 0  | 0 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 10     | 0  | 0  | 2 | 0 | 1  | 0 | 0 | 0  | 0 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
| 11     | 0  | 1  | 1 | 0 | 0  | 4 | 0 | 0  | 0 | 0 |
+--------+----+----+---+---+----+---+---+----+---+---+
On the training set, the algorithm made 934 mistakes for 1000 digits.
On the test set, the algorithm made 186 mistakes for 200 digits.
```