Neuronale Netze - Übung 5

Tobias Hahn
3073375

November 21, 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

```
1  import numpy as np
2  from texttable import Texttable
3
4  def clustering(digits, k):
5      dim_num = len(digits['data'][0])
6      digits_number = len(digits)
7      weight_vecs = np.random.rand(k, dim_num)
8      for i,v in enumerate(weight_vecs):
9          weight_vecs[i] = v / np.linalg.norm(v)
10
11     for i in range(0,100000):
12         current_instance = digits['data'][np.random.randint(0,digits_number)]
13         largest_index = -1
14         largest_value = float("-inf")
15
16         for j in range(0,k):
17             current_value = np.dot(current_instance, weight_vecs[j])
18             if (current_value > largest_value):
19                 largest_value = current_value
20                 largest_index = j
21
22         temp_vector = weight_vecs[largest_index] + current_instance
23         weight_vecs[largest_index] = temp_vector / np.linalg.norm(temp_vector)
24
25     return assign_numbers(digits, weight_vecs)
26
27 def assign_numbers(digits, weight_vecs):
28     count = np.zeros((len(weight_vecs), 10))
29
30     for digit in digits:
31         largest_index = -1
32         largest_value = float("-inf")
33
34         for j in range(0,len(weight_vecs)):
35             current_value = np.dot(digit['data'], weight_vecs[j])
36             if (current_value > largest_value):
37                 largest_value = current_value
38                 largest_index = j
39
40         count[largest_index][digit['value']] += 1
41
42     named_vecs = np.zeros(len(weight_vecs), dtype=[('vector', 'f', len(weight_vecs[0])), ('
           digit', 'i')])
43
44     for i,v in enumerate(named_vecs):
```

```
45          named_vecs['vector'][i] = weight_vecs[i]
46          named_vecs['digit'][i] = np.argmax(count[i])
47
48      print(named_vecs['digit'])
49
50      return named_vecs
51
52  def predict_number(digit, named_vecs):
53      largest_index = -1
54      largest_value = float("-inf")
55
56      for j in range(0,len(named_vecs)):
57          current_value = np.dot(digit, named_vecs['vector'][j])
58          if (current_value > largest_value):
59              largest_value = current_value
60              largest_index = j
61
62      return named_vecs['digit'][largest_index]
63
64  def calc_error(digits, named_vecs):
65      error = 0
66
67      for d in digits:
68          if (d['value'] != predict_number(d['data'], named_vecs)):
69              error += 1
70
71      return error
72
73
74  def print_results(digits, weight_vecs):
75      count = np.zeros((len(weight_vecs), 10))
76
77      for digit in digits:
78          largest_index = -1
79          largest_value = float("-inf")
80
81          for j in range(0,len(weight_vecs)):
82              current_value = np.dot(digit['data'], weight_vecs[j])
83              if (current_value > largest_value):
84                  largest_value = current_value
85                  largest_index = j
86
87          count[largest_index][digit['value']] += 1
88
89      t = Texttable()
90      t.add_row(["Vektor", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9"])
91      for i,v in enumerate(count):
92          t.add_row(np.insert(v, 0, i))
93      print t.draw()
```

../train.py

```
1   import numpy as np
2   import konkurrenz as ko
3
4   def extractDigits(filename, expected_num):
5       data_count = 0
6       digit_count = 0
7       data_points_per_digit = 192
8       data_points_per_line = 12
9
10      digits = np.zeros(expected_num, dtype=[('data', 'f', data_points_per_digit), ('value', 'i
            ')])
11
12      with open(filename) as f:
13          lines = f.readlines()
14
15      for i,line in enumerate(lines):
16          digits_line = line.split()
17          if (len(digits_line) == data_points_per_line):
18              for num in digits_line:
```

```
19                digits['data'][digit_count][data_count] = float(num)
20                data_count += 1
21            elif (len(digits_line) == 10):
22                for i,num in enumerate(digits_line):
23                    if (num == "1.0"):
24                        digits['value'][digit_count] = i
25                        break
26            else:
27                if (data_count == data_points_per_digit and digit_count < expected_num):
28                    digit_count += 1
29                    data_count = 0
30                else:
31                    print("Exited because of wrong data")
32                    raise SystemExit
33
34        if (digit_count == expected_num):
35            return digits
36        else:
37            print("Exited because of few digits")
38            raise SystemExit
39
40    if __name__ == "__main__":
41        training_name = "./data/digits.trn"
42        training_number = 1000
43        training_digits = extractDigits(training_name, training_number)
44
45        test_name = "./data/digits.tst"
46        test_number = 200
47        test_digits = extractDigits(test_name, test_number)
48
49        named_vectors = ko.clustering(training_digits, 10)
50        ko.print_results(training_digits, named_vectors['vector'])
51        print("On the training set, the algorithm made {0} mistakes for {1} digits.".format(ko.
              calc_error(training_digits, named_vectors), len(training_digits)))
52        print("On the test set, the algorithm made {0} mistakes for {1} digits.".format(ko.
              calc_error(test_digits, named_vectors), len(test_digits)))
53
54        named_vectors = ko.clustering(training_digits, 12)
55        ko.print_results(test_digits, named_vectors['vector'])
56        print("On the training set, the algorithm made {0} mistakes for {1} digits.".format(ko.
              calc_error(training_digits, named_vectors), len(training_digits)))
57        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

```
[0 5 8 7 1 1 4 2 3 7]
+--------+-----+---+---+---+---+---+---+---+---+
| Vektor | 0   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
+--------+-----+---+---+---+---+---+---+---+---+---+
| 0      | 120 | 0 | 3 | 3 | 0 | 3 | 24| 1 | 2 | 22|
+--------+-----+---+---+---+---+---+---+---+---+---+
| 1      | 0   | 0 | 0 | 1 | 0 | 19| 1 | 0 | 1 | 6 |
+--------+-----+---+---+---+---+---+---+---+---+---+
| 2      | 10  | 11| 18| 11| 16| 36| 36| 13| 48| 11|
+--------+-----+---+---+---+---+---+---+---+---+---+
| 3      | 0   | 0 | 4 | 4 | 3 | 0 | 0 | 25| 2 | 25|
+--------+-----+---+---+---+---+---+---+---+---+---+
| 4      | 1   | 39| 15| 37| 1 | 0 | 0 | 7 | 6 | 1 |
+--------+-----+---+---+---+---+---+---+---+---+---+
| 5      | 2   | 59| 1 | 2 | 3 | 0 | 0 | 9 | 4 | 10|
+--------+-----+---+---+---+---+---+---+---+---+---+
| 6      | 19  | 0 | 1 | 2 | 77| 5 | 11| 18| 2 | 36|
+--------+-----+---+---+---+---+---+---+---+---+---+
| 7      | 0   | 0 | 32| 25| 0 | 1 | 3 | 1 | 14| 13|
+--------+-----+---+---+---+---+---+---+---+---+---+
| 8      | 1   | 0 | 3 | 28| 0 | 2 | 0 | 0 | 0 | 1 |
+--------+-----+---+---+---+---+---+---+---+---+---+
| 9      | 0   | 0 | 5 | 0 | 0 | 0 | 0 | 24| 0 | 0 |
+--------+-----+---+---+---+---+---+---+---+---+---+
On the training set, the algorithm made 529 mistakes for 1000 digits.
```

```
On the test set, the algorithm made 93 mistakes for 200 digits.
[0 3 2 3 2 1 4 1 1 1 7 5]
+---------+----+---+----+----+----+----+---+---+---+---+
| Vektor  | 0  | 1 | 2  | 3  | 4  | 5  | 6 | 7 | 8 | 9 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 0       | 30 | 4 | 3  | 1  | 5  | 2  | 6 | 2 | 5 | 7 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 1       | 0  | 0 | 0  | 0  | 0  | 1  | 0 | 0 | 1 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 2       | 0  | 3 | 11 | 3  | 0  | 0  | 0 | 6 | 1 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 3       | 1  | 0 | 1  | 15 | 0  | 1  | 0 | 0 | 0 | 7 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 4       | 0  | 0 | 7  | 0  | 1  | 0  | 0 | 2 | 0 | 1 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 5       | 0  | 9 | 0  | 0  | 1  | 0  | 0 | 0 | 0 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 6       | 0  | 0 | 0  | 0  | 13 | 0  | 4 | 2 | 0 | 3 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 7       | 0  | 0 | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 8       | 0  | 2 | 2  | 1  | 1  | 0  | 0 | 0 | 1 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 9       | 0  | 3 | 0  | 0  | 2  | 0  | 2 | 2 | 0 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 10      | 1  | 0 | 0  | 0  | 0  | 0  | 0 | 6 | 0 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
| 11      | 1  | 0 | 1  | 3  | 0  | 10 | 1 | 0 | 2 | 0 |
+---------+----+---+----+----+----+----+---+---+---+---+
On the training set, the algorithm made 516 mistakes for 1000 digits.
On the test set, the algorithm made 94 mistakes for 200 digits.
```

2 Verständnisfrage

Tote Knoten entstehen in einem Netz, wenn einer der Vektoren nie ausgewählt wird da immer andere Vektoren ein größeres Vektorprodukt mit den Beispielen hat (also "näher" dranliegt). Dieser Prozess ist selbstverstärkend, da die Vektoren die Updates bekommen "näher" an die Daten rücken und daher in Zukunft eher bevorzugt werden.

Das Vorkommen von toten Knoten kann dadurch zumindest erschwert werden, dass man die Updateregel anpasst. Die normale Updateregel sieht so aus:

$$w_m = w_m + x_j$$

mit anschließender Normierung. Wie man sehen kann wird hier das ganze Beispiel verwendet, der Vektor rückt dementsprechend nahe an die Daten heran und hat dadurch zukünftig einen Vorteil gegenüber den anderen. Im folgenden werden drei Arten der Anpassung vorgestellt und warum sie tote Knoten vermeiden.

2.1 Lernrate

Die Idee der Lernrate ist, Beispiele immer nur bis zu einem gewissen Grade in das Update einfließen zu lassen. Dies bewirkt, dass sich Vektoren nicht zu sehr verbessern, und sich dadurch die Chance erhöht dass andere Vektoren beim nächsten Update ausgewählt werden. Die Updateregel sieht dann folgendermaßen aus:

$$w_m = w_m + \eta * x_j \text{ wobei } \eta \in (0, 1]$$

2.2 Differenz

Die Grundidee bei der Differenz ist wie bei der Lernrate, plus dass der Gewichtsvektor nur um den Unterschied zwischen dem Beispiel und dem Gewichtsvektor verbessert wird. Dies bewirkt eine weitere Verlangsamung der Konvergenz, mit den selben Effekten wie bei der Lernrate:

$$w_m = w_m + \eta * (x_j - w_m) \text{ wobei } \eta \in (0, 1]$$

2.3 Stapelverarbeitung

Die Stapelverarbeitung stellt keine neue Formel für das Update dar, sondern eine Methode, wie beliebige Updateregeln angewendet werden. Statt dass Update gleich auszuführen wird die Differenz für alle Vektoren für eine gewisse Anzahl an Updates akkumuliert, um erst danach angewandt zu werden. Die Idee dabei ist, dass die Vektoren nicht gleich verbessert werden, und so auch initial "schlechte" Vektoren die Chance erhalten, sich Updates zu finden. Erst nach ein paar Durchgängen werden die Vektoren verbessert, mit der Hoffnung dass sich schlechte Vektoren Updates gefunden haben und sich dementsprechend verbessern.

2.4 Zusammenfassung

Für alle Veränderungen der Vermeidung von toten Knoten lässt sich sagen dass sie tote Knoten nicht verhindern können. Sie sind alle darauf ausgerichtet relativ schlechten Vektoren größere Chancen zu geben, sich für manche Beispiele doch als passend zu erweisen und sich so zu verbessern. Sind jedoch initial Vektoren ausgewählt die einfach gar nicht zu den Daten passen werden sie immer noch niemals ausgewählt und erhalten deswegen keine Updates.