Neuronale Netze - Übung 7

Tobias Hahn
3073375

December 3, 2016

1 Neuronales Netz

1.1 Implementierung

Das Neuronale Netz habe ich nicht mit Rprop implementiert sondern einfach mit fixer Lernrate. Das liegt daran dass ich das Netz ursprünglich als reine Matrixmultiplikation implementiert habe, die also nicht Beispiel pro Beispiel durchgeht sondern immer gleich alle Beispiele als Beispielmatrix nimmt. Bei so einer Vorgehensweise ist die adaptive Lernratenanpassung nicht hilfreich, weswegen ich sie weggelassen habe. Sehe natürlich ein dass das Punkteabzüge gibt, aber alles nochmal umbauen war mir zu umständlich.

1.2 Code

Der Code des Netzes ist hier:

Im folgenden zuerst der Quellcode für die beiden Klassen, danach zwei Bilder der Ergebnisse.

../code/neural_network.py

```
1  import numpy as np
2
3  def sigmoid(X, Y=None, deriv=False):
4      if not deriv:
5          return 1 / (1 + np.exp(-X))
6      else:
7          return sigmoid(X)*(1 - sigmoid(X))
8
9  def error(X, Y):
10     return 0.5 * (X - Y)**2
11
12 class Layer:
13     def __init__(self, size=None, is_input=False, is_error=False, is_error_sum=False,
           activation=sigmoid):
14         self.is_input = is_input
15         self.is_error = is_error
16         self.is_error_sum = is_error_sum
17
18         self.activation = activation
19
20         # Z is the matrix holding the activation values
21         self.Z = None
22         # W is the outgoing weight matrix for this layer
23         self.W = None
24         # S is the matrix that holds the inputs to this layer
25         self.S = None
26         # D is the matrix that holds the deltas for this layer
27         self.D = None
28         # Y is the matrix holding the true labels
29         self.Y = None
30         # Fp is the matrix that holds the derivatives of the activation function
31         self.Fp = None
32
33         if not is_error and not is_error_sum:
34             self.size = size[0]-1
35             self.W = np.random.normal(size=size, scale=1E-4)
36
37     def forward_propagate(self):
38         if self.is_input:
39             return self.Z.dot(self.W)
40
41         if self.is_error_sum:
42             return np.sum(self.S)
43
44         if self.is_error:
45             self.S = sigmoid(self.S)
46             self.D = (self.S - self.Y).T
47
48         self.Z = self.activation(self.S, self.Y)
```

```
49
50          if self.is_error:
51              return self.Z
52
53          self.Fp = self.activation(self.S, deriv=True).T
54          # For hidden layers, we add the bias values here
55          self.Z = np.append(self.Z, np.ones((self.Z.shape[0], 1)), axis=1)
56          return self.Z.dot(self.W)
57
58      def set_training_values(self, Y):
59          self.Y = Y
60
61      def set_training_data(self, X):
62          self.Z = np.append(X, np.ones((X.shape[0], 1)), axis=1)
63
64
65  class NN:
66      def __init__(self, layer_config):
67          self.layers = []
68          self.num_layers = len(layer_config)
69
70          for i in range(self.num_layers-1):
71              if i == 0:
72                  #addition of bias unit to input layer
73                  self.layers.append(Layer([layer_config[i]+1, layer_config[i+1]], is_input=True))
74              else:
75                  #addition of bias unit to hidden layer(s)
76                  self.layers.append(Layer([layer_config[i]+1, layer_config[i+1]]))
77          #error layer (error for every output neutron)
78          self.layers.append(Layer(is_error=True, activation=error))
79          #error sum layer (just sum of all errors)
80          self.layers.append(Layer(is_error_sum=True))
81
82          self.num_layers = len(self.layers)
83          #our error layer, important for backpropagation
84          self.error_layer = self.num_layers - 2
85
86      def forward_propagate(self, X, Y):
87          # initialize input and true data
88          self.layers[0].set_training_data(X)
89          self.layers[self.error_layer].set_training_values(Y)
90
91          #forward propagation
92          for i in range(self.num_layers-1):
93              self.layers[i+1].S = self.layers[i].forward_propagate()
94          return self.layers[-1].forward_propagate()
95
96      def train(self, X, Y, eta):
97          error = self.forward_propagate(X, Y)
98
99          #backward propagation
100         for i in range(self.error_layer-1, 0, -1):
101             # We do not calculate deltas for the bias values
102             W_nobias = self.layers[i].W[0:-1, :]
103
104             self.layers[i].D = W_nobias.dot(self.layers[i+1].D) * self.layers[i].Fp
105
106         #weight update
107         for i in range(0, self.error_layer):
108             W_grad = -eta*(self.layers[i+1].D.dot(self.layers[i].Z)).T
109             self.layers[i].W += W_grad
110
111         #return the error as a benchmark
112         return error
113
114     def evaluate(self, train_data, train_labels, test_data, test_labels, difference=0.001,
            eta=0.001):
115         last_error = float("inf")
116         while True:
117             error = self.train(train_data, train_labels, eta)
118             if (error < last_error and last_error - error < difference):
```

```
119                 break
120             last_error = error
121
122         test_error = self.forward_propagate(test_data, test_labels)
123         print("Anzahl verdeckter Knoten: {0} / Training error: {1:.5f} / Test error: {2:.5f}".
                format(self.layers[1].size, last_error, test_error))
```

../code/train.py

```
1  import numpy as np
2  from neural_network import NN
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', 'f
           ', 10)])
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             digits['value'][digit_count] = np.zeros(10)
23             for i,num in enumerate(digits_line):
24                 if (num == "1.0"):
25                     digits['value'][digit_count][i] = float(num)
26                     break
27         else:
28             if (data_count == data_points_per_digit and digit_count < expected_num):
29                 digit_count += 1
30                 data_count = 0
31             else:
32                 print("Exited because of wrong data")
33                 raise SystemExit
34
35     if (digit_count == expected_num):
36         return digits
37     else:
38         print("Exited because of few digits")
39         raise SystemExit
40
41 if __name__ == "__main__":
42     train_name = "../data/digits.trn"
43     train_number = 1000
44     train_digits = extractDigits(train_name, train_number)
45
46     test_name = "../data/digits.tst"
47     test_number = 200
48     test_digits = extractDigits(test_name, test_number)
49
50     for i in range(0,11):
51         nn = NN([192, i*10, 10])
52         nn.evaluate(train_digits['data'], train_digits['value'], test_digits['data'],
               test_digits['value'])
```

1.3 Ergebnisse

```
1  Anzahl verdeckter Knoten: 0 / Training error: 444.43581 / Test error: 89.10453
```

```
Anzahl verdeckter Knoten: 10 / Training error: 1.48327 / Test error: 16.24645
Anzahl verdeckter Knoten: 20 / Training error: 445.31566 / Test error: 89.30035
Anzahl verdeckter Knoten: 30 / Training error: 445.38601 / Test error: 89.24764
Anzahl verdeckter Knoten: 40 / Training error: 0.57803 / Test error: 10.83979
Anzahl verdeckter Knoten: 50 / Training error: 0.54577 / Test error: 11.64844
Anzahl verdeckter Knoten: 60 / Training error: 0.51040 / Test error: 11.97264
Anzahl verdeckter Knoten: 70 / Training error: 0.51243 / Test error: 11.41059
Anzahl verdeckter Knoten: 80 / Training error: 0.49254 / Test error: 11.84091
Anzahl verdeckter Knoten: 90 / Training error: 0.50322 / Test error: 12.59262
Anzahl verdeckter Knoten: 100 / Training error: 0.49312 / Test error: 11.94842
```

1.4 Interpretation

Wie man sieht ist der Error mit einer Anzahl verdeckter Knoten unter 30 recht hoch, außer bei 10. Dies kann daran liegen dass für diese Beispiele die Lernrate zu niedrig gewählt wurde, jedoch auch daran (das wäre der normale Fehler bei zu wenigen Knoten) die Knoten nicht ausreichen um alle relevanten Features der Bilder abzubilden. Über 30 verändert sich dann der Fehler nicht mehr so stark, was wohl daran liegt dass es keine interessanten Features mehr gibt die die Fehlerrate verbessern. Dass bei 10 die Lernrate so niedrig liegt liegt einfach daran dass die verdeckte Schicht wahrscheinlich als Output-Schicht agiert, also jeweils den Erwartungswert einer Ziffer abbildet.