Projektseminar - Geometrische Datenanalyse

Präsentation der Ergebnisse

Teppe Marius

05. Februar 2020

Der Rahmen dieser Präsentation:

Studiengang	:	Master Angewandte Mathematik
Modul	:	Projektseminar
Modulname	:	Geometrische Datenanalyse
Professor	:	Jan Phillip Hoffmann

Agenda

1 Einleitung

2 Punkte generieren

3 Geometrie erkennen

4 Neuronale Netze

5 Aussicht

Was ist grundsätzlich das Ziel?

Betrachtet werden Streudiagramme oder Punktewolken von statistischen Daten.

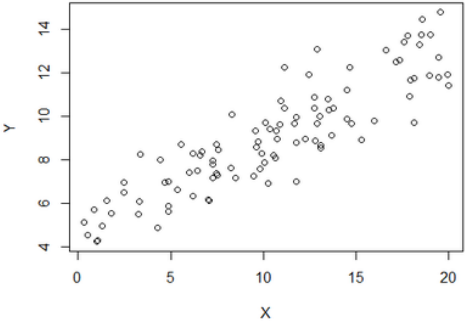


Abbildung: Streudiagramm [2]

Betrachtet werden Streudiagramme oder Punktewolken, die eine Geometrie bilden.

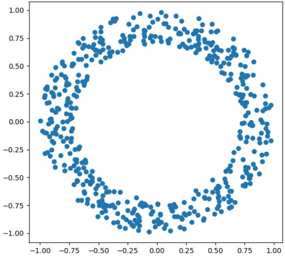


Abbildung: Kreisring

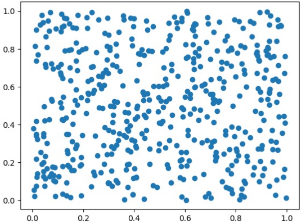


Abbildung: Quadrat

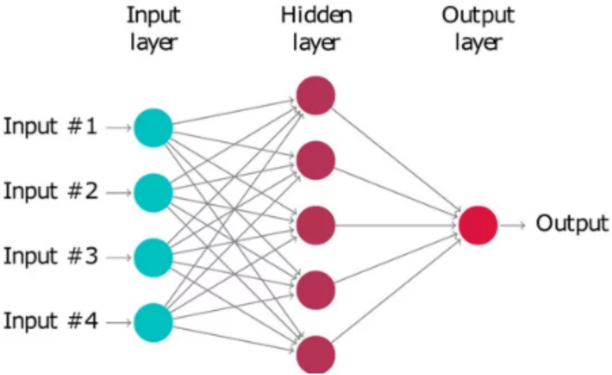
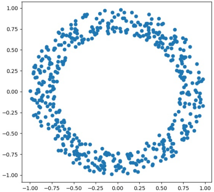
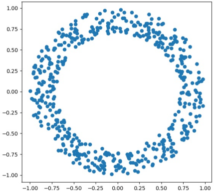


Abbildung: Neuronales Netz [1]

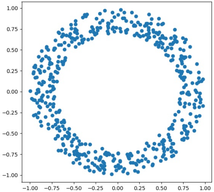
Das Neuronale Netz erkennt eine bestimmte Geometrie, wenn es darauf trainiert ist.

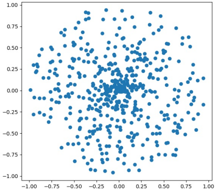
Aber ist das *Hidden Layer* Geometrie-erhaltend?





Kreisring $\Rightarrow$ Bleibt Kreisring





Kreisring $\Rightarrow$ Andere Geometrie

Agenda

1 Einleitung

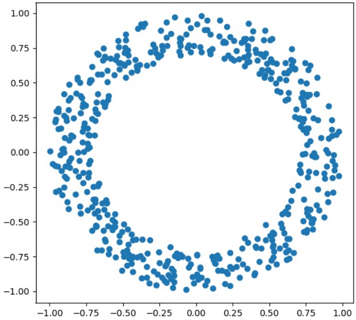
2 Punkte generieren

3 Geometrie erkennen

4 Neuronale Netze

5 Aussicht

In dieser Arbeit werden Punktewolken in Form von Kreisringen betrachtet.



Der Python-Code für diese Abbildung:

```
import numpy as np
import matplotlib.pyplot as plt

##### eingabe #####
innen = 0.7
aussen = 1
pi = np.pi
num_samples = 500

##### verarbeitung #####
points = np.zeros((num_samples,2))
phi = np.random.uniform(0, 2*pi, num_samples)
r = np.random.uniform(innen, aussen, num_samples)
x = r * np.cos(phi)
y = r * np.sin(phi)
points = np.vstack((x,y)).T

##### ausgabe #####
fig = plt.figure()
ax = fig.gca()
ax.plot(points[:,0],points[:,1],'o')
```

Agenda

1 Einleitung

2 Punkte generieren

3 Geometrie erkennen

4 Neuronale Netze

5 Aussicht

Definition: Simplex

Sei $k \in \mathbb{N}$ und seien $v_0, \ldots, v_k$ affin unabhängige Punkte des $\mathbb{R}^n$ (oder eines "n"-dimensionalen Vektorraums über $\mathbb{R}$) gegeben, so ist das "von $v_0, \ldots, v_k$ aufgespannte (oder erzeugte) Simplex" Δ gleich folgender Menge:

$$\Delta = \left\{ x \in \mathbb{R}^n : x = \sum_{i=0}^{k} t_i v_i \text{ mit } 0 \leq t_i \leq 1 \text{ und } \sum_{i=0}^{k} t_i = 1 \right\}$$

[3]

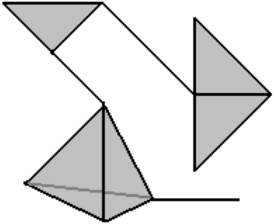
Bild eines Simplex:

[3]

Definition: Simplizialkomplex

Ein "geometrischer Simplizialkomplex" $\mathcal{S}$ ist eine Menge von Simplizes in einem euklidischen Raum $\mathbb{R}^d$ mit der Eigenschaft, dass jede Facette $\sigma' \subseteq \sigma$ eines Simplexes $\mathcal{S}$ wieder zu $\mathcal{S}$ gehört und dass für alle Simplizes $\sigma, \tau \in \mathcal{S}$ der Durchschnitt $\sigma \cap \tau$ entweder leer oder eine gemeinsame Facette von σ und τ ist. Mit $|\mathcal{S}|$ wird die Vereinigung aller Simplizes des geometrischen Komplexes bezeichnet. [4]

Bild eines Simplizialkomplex:



[4]

Definition: Vietoris-Rips Komplex

In der Topologie bezeichnet das "Vietoris-Rips Komplex", auch nur "Rips Komplex" genannt, einen Simplizialkomplex, der über einen metrischen Raum $\mathcal{M}$ und einer Distanz ε gebildet wird, indem jede endliche Menge von Punkten, deren Abstand kleiner als ε ist, zu einem Simplex geformt werden.

Der Python-Code für das Rips-Komplex

```python
import gudhi

##### rips complex erstellen
def rips(points, epsilon):
    rips_complex = gudhi.RipsComplex(points = points, max_edge_length=epsilon)
    simplex_tree = rips_complex.create_simplex_tree(max_dimension=2)
    return simplex_tree
```

Der Python-Code für das Simplizialkomplex

```
In [17]: simplex_tree = rips(points, epsilon)

In [18]: simplex_tree.get_filtration()
```

Der Python-Code des Simplizialkomplex mit 10 Punkten

```
Out[26]:
[([0], 0.0),
 ([1], 0.0),
 ([2], 0.0),
 ([3], 0.0),
 ([4], 0.0),
 ([5], 0.0),
 ([6], 0.0),
 ([7], 0.0),
 ([8], 0.0),
 ([9], 0.0),
 ([2, 6], 0.18536993855549998),
 ([5, 7], 0.2491465184947633),
 ([2, 3], 0.32084609610551607),
 ([1, 9], 0.376447919863614),
 ([1, 3], 0.46820704718824946),
 ([5, 8], 0.4973156629813336),
 ([3, 6], 0.504004646410561),
 ([2, 3, 6], 0.504004646410561),
 ([6, 8], 0.5613996822927588),
 ([2, 8], 0.6886169917575494),
 ([2, 6, 8], 0.6886169917575494)]
```

Definition: Eulercharakteristik

Die Eulercharakteristik, auch Euler-Poincare Charakteristik genannt, ist im mathematischen Gebiet der Topologie eine Kennzahl für topologische Räume.
Die Eulercharakteristik χ berechnet sich aus der endlichen alternierenden Summe über B_i, wobei B_i die anzahl der Elemente unendlicher Ordnung in einer Basis einer endlich erzeugten abelschen Gruppe ist.

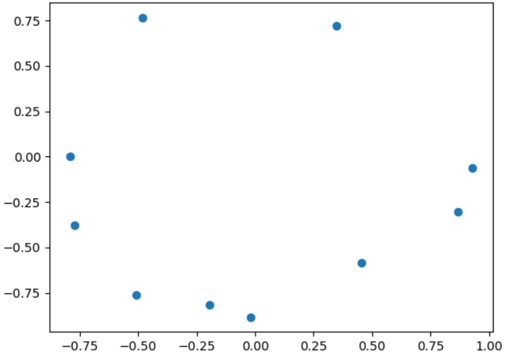
$$\chi = \sum_{i=0}^{n} (-1)^i f_i \ ,$$

wobei f_i die Anzahl der i-dimensionalen Simplizes darstellt.

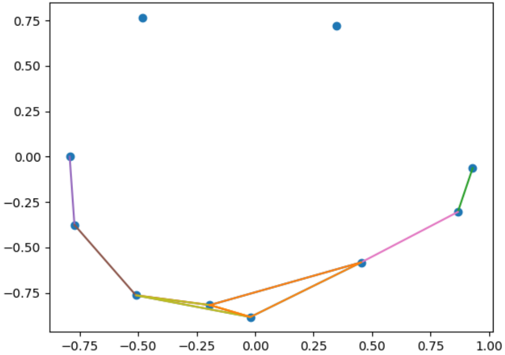
Der Python-Code für die Eulercharakteristik

```
def euler(points, epsilon):
    simplex_tree = rips(points, epsilon)
    a = simplex_tree.get_filtration()

    count_areas = 0
    count_edges = 0
    for i in range(np.shape(a)[0]):
        if np.shape(a[i][0])[0] == 3:
            count_areas += 1
            if areas_plotten == 1:
                points_area = np.zeros((4,2))
                points_area[0:3] = points[a[i][0]]
                points_area[3] = points_area[0]
                plt.plot(points_area[:,0],points_area[:,1])
        if np.shape(a[i][0])[0] == 2:
            count_edges += 1
            if edges_plotten == 1:
                points_edges = np.zeros((2,2))
                points_edges = points[a[i][0]]
                plt.plot(points_edges[:,0],points_edges[:,1])

    ecken = num_samples
    kanten = count_edges
    flaechen = count_areas

    euler_characteristic = ecken - kanten + flaechen

    return euler characteristics
```

Die 10 Punkte für das Simplizialkomplex



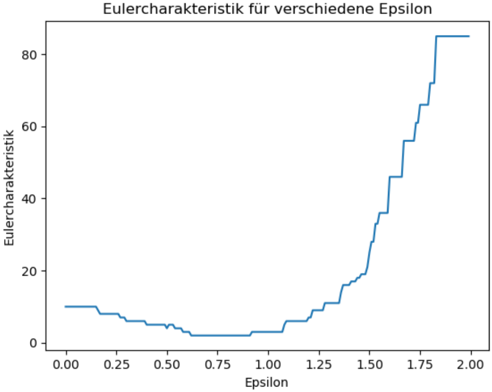
Das Simplizialkomplex für $\varepsilon = 0.7$



Damit ergibt sich die Eulercharakteristik:

$$\chi = Ecken - Kanten + Flchen = 10 - 9 + 2 = 3$$

Verschiedene Eulercharakteristiken in Abhängigkeit von ε



Agenda

1 Einleitung

2 Punkte generieren

3 Geometrie erkennen

4 Neuronale Netze

5 Aussicht

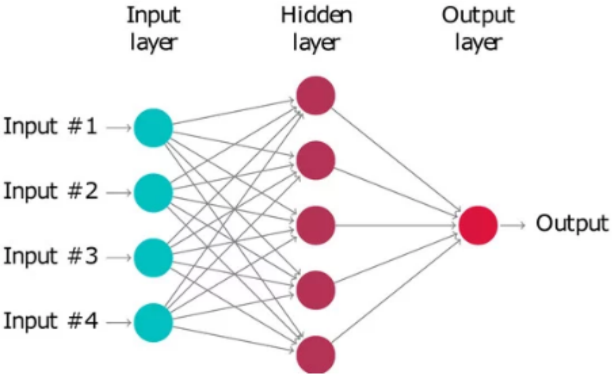


Abbildung: Neuronales Netz [1]

Zur Grundsätzlichen Auffassung von NN

Neuronale Netze können aufgefasst werden als:

- Gewichteter mathematischer Graph
- Knotenpunkte
- Kanten (gewichtet und gerichtet)

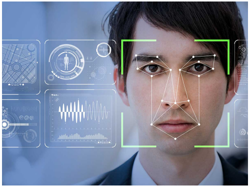
Sie bestehen aus:

- Eingangsvariablen
- Verarbeitung
- Ausgangsvariable

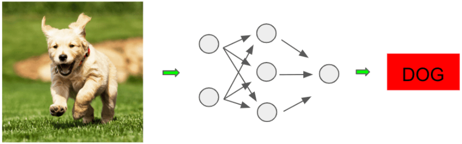
Zur Grundsätzlichen Auffassung von NN

Neuronale Netze werden genutzt für:

- Texterkennung
- Bilderkennung
- Gesichtserkennung



Gesichtserkennung [5]



Das ist ein Hund[1]

Der Aufbau des NN

- Welche Gestalt haben die Eingangsvariablen?
- Wie viele *Hidden Layer* gibt es?
- Welche Aktivierungsfunktion wird genutzt?

Die zweidimensionalen Eingangsvariablen werden zu einem Vektor transformiert.

$$\begin{array}{c|c} x & y \\ \hline x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \\ x_5 & y_5 \\ \ldots & \ldots \end{array} \quad \rightarrow \quad \begin{pmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ x_3 \\ y_3 \\ x_4 \\ y_4 \\ x_5 \\ y_5 \\ \ldots \end{pmatrix}$$

- In dieser Arbeit wurde sich auf ein einzelnes *Hidden Layer* beschränkt.
- Die gewählte Aktivierungsfunktion ist die logistische Funktion (auch *Sigmoid Function* genannt)

$$\mathrm{sig}(x) = \frac{1}{1+\exp^{-x}}$$

Die drei Schritte eines Neuronalen Netzes lauten:

1. Aufbau
2. Training
3. Test

Der Code für die Klasse des NN

```python
def sigmoid(x):
    return 1.0/(1+ np.exp(-x))

def sigmoid_derivative(x):
    return sigmoid(x)*(1.0-sigmoid(x))
#    return x * (1.0 - x)

def compute_loss(y_hat, y):
    return ((y_hat - y)**2).sum()

class NeuralNetwork:
    def __init__(self, x, y):
        self.input      = x
        self.weights1   = np.random.rand(self.input.shape[1],2*training_anzahl)
        self.weights2   = np.random.rand(2*training_anzahl,1)
        self.y          = y
        self.output     = np.zeros(self.y.shape)

    def feedforward(self):
        self.layer1 = sigmoid(np.dot(self.input, self.weights1))
        self.output = sigmoid(np.dot(self.layer1, self.weights2))

    def backprop(self):
        # application of the chain rule to find derivative of the loss function with respect to weights2 and weights1
        d_weights2 = np.dot(self.layer1.T, (2*(self.y - self.output) * sigmoid_derivative(self.output)))
        d_weights1 = np.dot(self.input.T,  (np.dot(2*(self.y - self.output) * sigmoid_derivative(self.output), self.weights2.T) * sigmoid_derivative(self.layer1)))

        # update the weights with the derivative (slope) of the loss function
        self.weights1 += d_weights1
        self.weights2 += d_weights2
```

Der Code für das Trainieren des NN

```
nn = NeuralNetwork(X,y)

loss_values = []

for i in range(10000):
    nn.feedforward()
    nn.backprop()
    loss = compute_loss(nn.output, y)
    loss_values.append(loss)
```

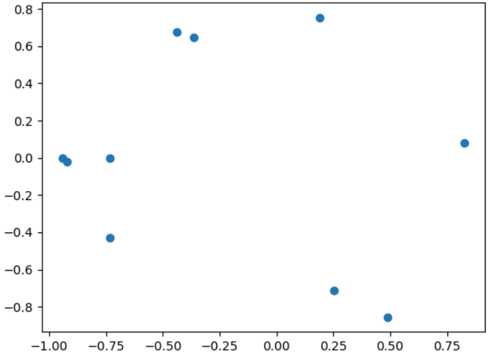
Der Code zum Testen des NN

```
phi = np.random.uniform(0, 2*pi, num_samples)
r = np.random.uniform(innen, aussen, num_samples)

x_test = r * np.cos(phi)
y_test = r * np.sin(phi)
points = np.vstack((x_test,y_test)).T
punkte = points.reshape(1,num_samples*2)
l1 = sigmoid(np.dot(punkte, nn.weights1))
output = sigmoid(np.dot(l1, nn.weights2))
```

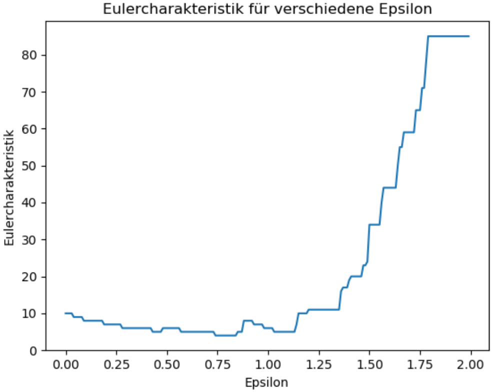
Der Code für das Testergebnis des NN

```
In [54]: phi = np.random.uniform(0, 2*pi, num_samples)
    ...: r = np.random.uniform(innen, aussen, num_samples)
    ...:
    ...: x_test = r * np.cos(phi)
    ...: y_test = r * np.sin(phi)
    ...: points = np.vstack((x_test,y_test)).T
    ...: punkte = points.reshape(1,num_samples*2)
    ...: l1 = sigmoid(np.dot(punkte, nn.weights1))
    ...: output = sigmoid(np.dot(l1, nn.weights2))
    ...:
    ...: print('Es soll 1 sein, tatsächlich ist es ',output[0,0])
Es soll 1 sein, tatsächlich ist es  0.7835177422637543
```

Plot der Testpunkte



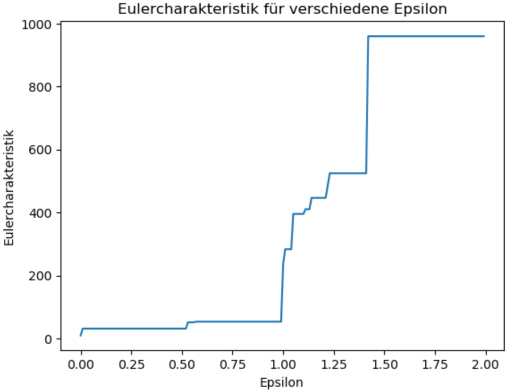
Eulercharakteristik der Testpunkte



Eulercharakteristik der Punkte im *Hidden Layer*

$$hidden\_points = sigmoid(punkte, nn.weights1)$$

Eulercharakteristik der Punkte im *Hidden Layer*



Agenda

1 Einleitung

2 Punkte generieren

3 Geometrie erkennen

4 Neuronale Netze

5 Aussicht

Verbesserungen

- Eulercharakteristik sollte nicht wachsen, sondern auf eins sinken
- Trainingsdaten besser aggregieren

Alternativen

- Einlesen einzelner Punkte in das Neuronale Netzwerk
- Andere Aktivierungsfunktion
- Vergleich zwischen Čech- und Rips-Komplex

That's it

Quellen

- Jäger, George (2019), *Replacing Rules by Neural Networks A Framework for Agent-Based Modelling*, University of Graz
- Smigierski, Jakob (Stand 25.01.2020), Statistik-Beratung: Streudiagramm mit R
- Wikipedia Simplex, `https://de.wikipedia.org/wiki/Simplex_(Mathematik)?veaction=edit&section=3`, Stand 25.01.2020
- Wikipedia Simplizialkomplex, `https://de.wikipedia.org/wiki/Simplizialkomplex`, Stand 25.01.2020
- Nicholls Jones, Sophie (06.05.2018), CPA Canada, *Breaking down biometrics, from palm prints to facial recognition to vein scans*, `https://www.cpacanada.ca/en/news/innovation/2018-06-05-breaking-down-biometrics`

Quellen

(04.2019), *Deep Dive Into Neural Networks*,
https://www.mihaileric.com/posts/deep-dive-into-neural-networks/