Unsupervised Deep Embedding for Clustering Analysis

Bachelorseminar Data Mining

Lukas Mahr

Ludwig-Maximilians-Universität München

Unsupervised Deep Embedding for Clustering Analysis
Bachelorseminar Data Mining
Lukas Mahr
Ludwig-Maximilians-Universität München

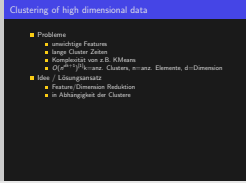
Roadmap

1. Clustering of high dimensional data
2. Einleitung zu Neuronalen Netzen
   - Idee
   - Künstliches Neuron
   - Layer/Schicht
   - Aktivierungsfunktion
   - Loss/Kostenfunktion
   - Backpropagation mit Gradient descent
3. Autoencoders
   - Idee
   - Aufbau
4. Stecked Autoencoders
   - Idee
   - Aufbau

Roadmap
Clustering of high dimensional data
Einleitung zu Neuronalen Netzen
Idee
Künstliches Neuron
Layer/Schicht
Aktivierungsfunktion
Loss/Kostenfunktion
Backpropagation mit Gradient descent
Autoencoders
Idee
Aufbau
Stecked Autoencoders
Idee
Aufbau

Clustering of high dimensional data

- Probleme
  - unwichtige Features
  - lange Cluster Zeiten
  - Komplexität von z.B. KMeans
  - $O(n^{dk+1})$[1] k=anz. Clusters, n=anz. Elemente, d=Dimension
- Idee / Lösungsansatz
  - Feature/Dimension Reduktion
  - in Abhängigkeit der Clustere



viele Daten Punkte viele Distanzen zu berechnen schwierig zu visualisieren ohne die Dimensionen zu reduzieren Komplexität von Kmeans die exponentiell ansteigt

Einleitung zu Neuronalen Netzen

- Nicht-lineare statistische Modelle zur Informationsverarbeitung
- Informationsverarbeitung umfasst hierbei unter anderem
  - Klassifikation
  - Prognosenerstellung
- Units der Neuronalen Netze angelehnt an Neuronen
  - Inputs zusammenfassen
  - Mit Schwellenwert vergleichen bzw. aktivieren
- Verbindungen zwischen Units angelehnt an Synapsen
  - Gewichtung mit verstärkender oder schwächender Wirkung

Einleitung zu Neuronalen Netzen
Nicht-lineare statistische Modelle zur Informationsverarbeitung
Informationsverarbeitung umfasst hierbei unter anderem
Klassifikation
Prognosenerstellung
Units der Neuronalen Netze angelehnt an Neuronen
Inputs zusammenfassen
Mit Schwellenwert vergleichen bzw. aktivieren
Verbindungen zwischen Units angelehnt an Synapsen
Gewichtung mit verstärkender oder schwächender Wirkung

wofür braucht man neuronale Netze
- Klassifizierung von Daten
- Prognose eines bestimmten Wertes
- Neuronen an Neuronen in unserem Gehirn angelehnt
- Verbindungen zwischen Künstlichen Neuronen an Synapsen angelehnt
- supervised learning = überwachtes lernen
– man besitzt Daten mit Beschriftungen (labels)
- unsperviesed learning nicht überwachtes lernen
– unbeschriftete Daten, also es sind keine labels vorhanden

Einleitung zu Neuronalen Netzen

Künstlichen Neurons

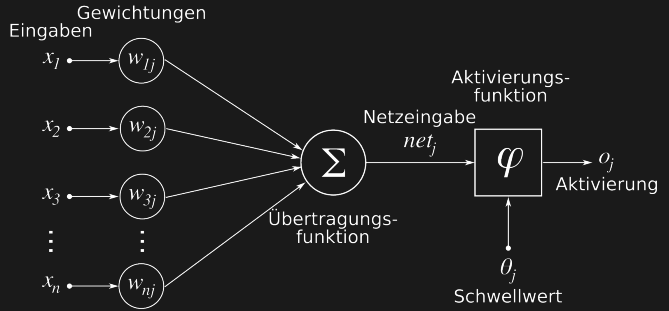
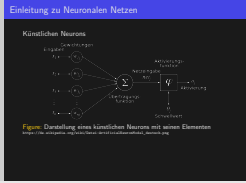


Figure: Darstellung eines künstlichen Neurons mit seinen Elementen
https://de.wikipedia.org/wiki/Datei:ArtificialNeuronModel_deutsch.png



$x_1, ..., x_n$ sind die input variablen, jede der Eingabe variablen besitzt ein Gewicht, $w_{1j}, ..., w_{nj}$. Diese werden Multipliziert und davon dann die summe berechnet. Hier die Übertragungsfunktion. Dazu wird ein Bias, in dem Fall der Schwellenwert gerechnet. Als letztes gibt es noch die Aktivierungsfunktion die meistens einen Wert zwischen 0 und 1 zurückgibt. Das ist dann der input für das nächste Neuron.

Einleitung zu Neuronalen Netzen

Layer/Schichten

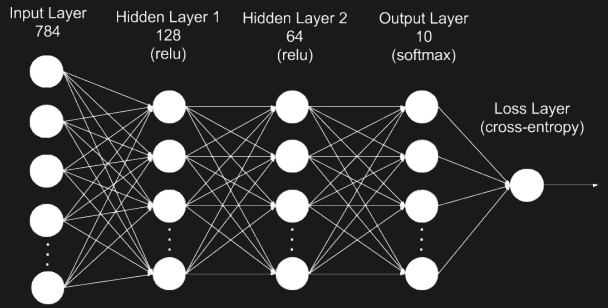
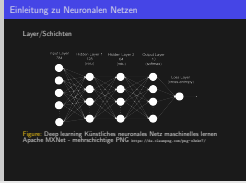


Figure: Deep learning Künstliches neuronales Netz maschinelles lernen Apache MXNet - mehrschichtige PNG https://de.cleanpng.com/png-x3zkr7/



Layer/Schicht sind mehrere Neuronen die mit allen Neuronen des nächsten Layer/Schicht verbunden sind. Alle Neuronen in einem Layer haben die gleiche Aktivierungsfunktion. Hidden Layer haben meistens die Aktivierungsfunktion rectified linear, da diese recht einfach und schnell zu berchnen ist. Das outputlayer hat meistens eine etwas kompliziertere Funktion wie softmax oder sigmoid. Abhängig von der Aufgabe des Netzwerkes. Letztes Layer hier direkt mit dem Loss

Einleitung zu Neuronalen Netzen

Aktivierungsfunktionen

Figure: Rectifier-Aktivierungsfunktion
https://de.wikipedia.org/wiki/Datei:Activation_rectified_linear.svg

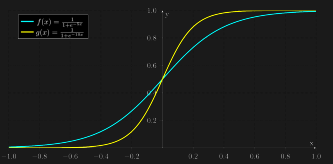
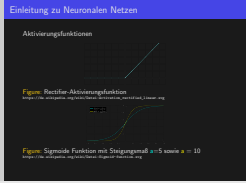


Figure: Sigmoide Funktion mit Steigungsmaß a=5 sowie a = 10
https://de.wikipedia.org/wiki/Datei:Sigmoid-function.svg



alles negativ ist wird bei relu zu 0 während bei sigmoid, abhängig von der Steigung Werte zwischen -1 und 1 möglich sind

Einleitung zu Neuronalen Netzen

Loss/Kostenfunktion

Mean Squared Error

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

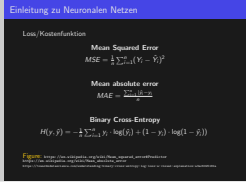
Mean absolute error

$$MAE = \frac{\sum_{i=1}^{n}|\hat{y}_i - y_i}{n}$$

Binary Cross-Entropy

$$H(y, \hat{y}) = -\frac{1}{n}\sum_{i=1}^{n} y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i))$$

Figure: https://en.wikipedia.org/wiki/Mean_squared_error#Predictor
https://en.wikipedia.org/wiki/Mean_absolute_error
https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a



Man berechnet immer den unterschied zwischen den wahren labeln und den predicteden labeln um zu erkenne wie weit diese auseinander liegen. Es wird immer versucht den Loss zu minimieren. Also ein Minimum der Kostenfunktion zu finden. Die Parameter der Funktion, welche angepasst werden müssen sind alle weights und biases der einzelnen Neuronen und den Layern.

Backpropagation mit Gradient descent

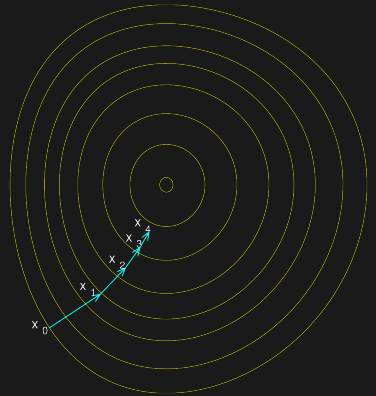
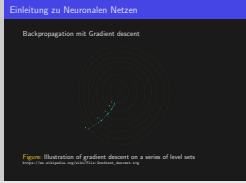


Figure: Illustration of gradient descent on a series of level sets
https://en.wikipedia.org/wiki/File:Gradient_descent.svg



Über Backpropagation wird hier mit z.B Gradient Descent die Loass funktion minimiert. Der Gradient der Loss/ Kostenfunktion wird für alle wigths and biases gleichzeitig berechnet. Man kann sich das vorstellen, wie eine Kugel die man einen in einer Hügellandschaft rollen lässt ein kleinen schritten und zwischen den schritten immer nach der Steigung des Abhanges schaut und dabei versucht die Kugel in das tiefste Tal zu bekommen.

Idee

- Der Input ist der Output
- Engpass in der Mitte

Autoencoders
Idee
- Der Input ist der Output
- Engpass in der Mitte

Ein Autoencoder ist ein feedforward Neural network, was versucht den input zu Kopieren. Das hört sich im ersten Moment nutzlos an, hat aber doch ein paar Anwendungsmöglichkeiten. Dazu gehört z.B Denosing oder Reduktion des Features Spaces

Autoencoders

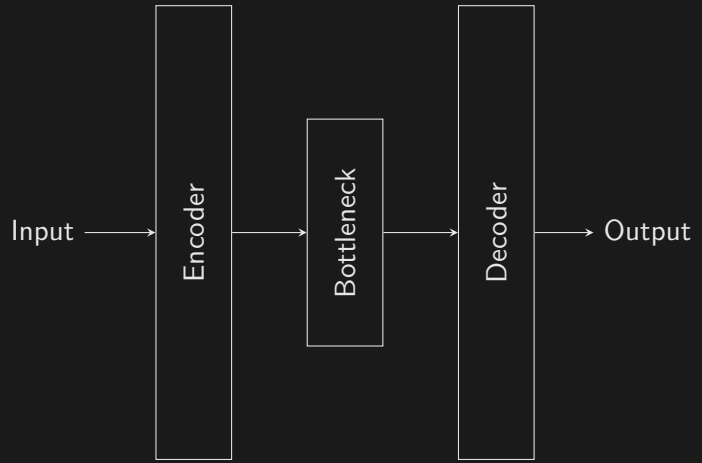
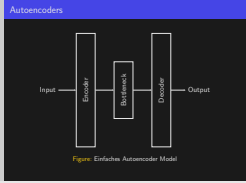


Figure: Einfaches Autoencoder Model



- Aufbau, Das Netzwerk besteht im Grunde aus 2 Teilen.
- Encoder und Decoder
- Encoder Transformiert die Input Daten in eine kleinere gewünschte Dimension
- Decoder Transformiert die Daten aus der Kleinen Dimension zurück in die Orginal Dimensionen.
- Hoffnung das der Encoder die Daten auf die Wichtigsten Features reduziert.

Autoencoders

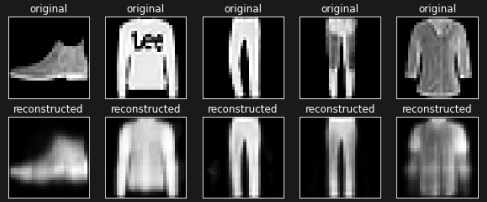


Figure: Orginal und Decoded Bilder

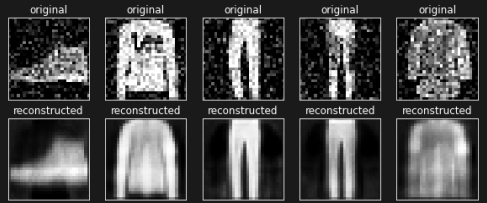
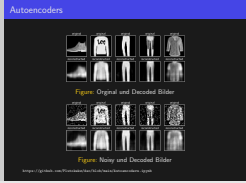


Figure: Noisy und Decoded Bilder

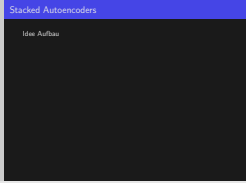
https://github.com/Plutokekz/dec/blob/main/Autoencoders.ipynb



- 784 input zu 10 zu 784 (28*28)
- Bottleneck size = 10 oder feature redutktion auf 10
- Random noise

Stacked Autoencoders

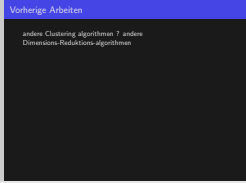
Idee Aufbau



Stacked Autoincoders ist genau das was der name Sagt.
mehrere Autoencoder hintereinander.
man started mit einem Autoencoder, den man Trainniert.
dann entfernt man den decoder Teil
und setzt an diese stelle den nächsten Autoencoder, wird wieder trainniert bis die gewünschte anzahl der stacks erreicht ist.
Besonderheit man trainiert die folgenden Autoencoder mit dem Output des vorigen encoders.

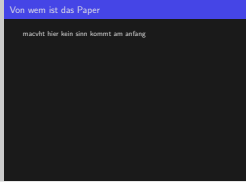
Vorherige Arbeiten

andere Clustering algorithmen ? andere Dimensions-Reduktions-algorithmen



Von wem ist das Paper

macvht hier kein sinn kommt am anfang



Referenzen

k-means clustering
https://en.wikipedia.org/wiki/K-means_clustering#Complexity

Referenzen
k-means clustering
https://en.wikipedia.org/wiki/K-means_clustering#Complexity