Intelligente Systeme

- Skript -

Prüfungsform: Referat

Adrian Helberg
Matr.Nr. 2309051

Abgabe: 18.02.2020

Zusammenfassung

Diese Arbeit beschäftigt sich mit Konzepten der vier Kernbereiche aus *intelligente Systeme*: **Suchen**, **Lernen**, **Verarbeitung von Sequenzen** und **Ethik**. Zum einen werden Problemstellungen aufgezeigt, zum anderen auf Implementationen verwiesen, die in der zugehörigen Präsentation vorgestellt werden.
Eine Lösungsstrategie für das *Problem des Handlungsreisenden (Traveling Salesman Problem)* wird im Kapitel **Suchen** beschrieben. *Selbstorganisierte Karten (Selforganizing Maps)* zeigen eine Strategie des **Lernen**s, ein *Deep Learning*- Ansatz zur Datenanalyse von Sensordaten mobiler Endgeräte gibt Einblicke in die **Sequenzverarbeitung** und eine Diskussion zum *Trolley Problem* lädt zur Diskussion über die **Ethik** in Verbindung mit *intelligenten Systemen* ein.

Inhalt

1 Suchen

Dieses Kapitel beschäftigt sich mit der Umsetzungen einer Lösungsstrategie für das *Problem des Handlungsreisenden (Traveling Salesman Problem)* mithilfe einer informierten Suche. Im speziellen eine Umsetzung mit einem genetischen Algorithmus.

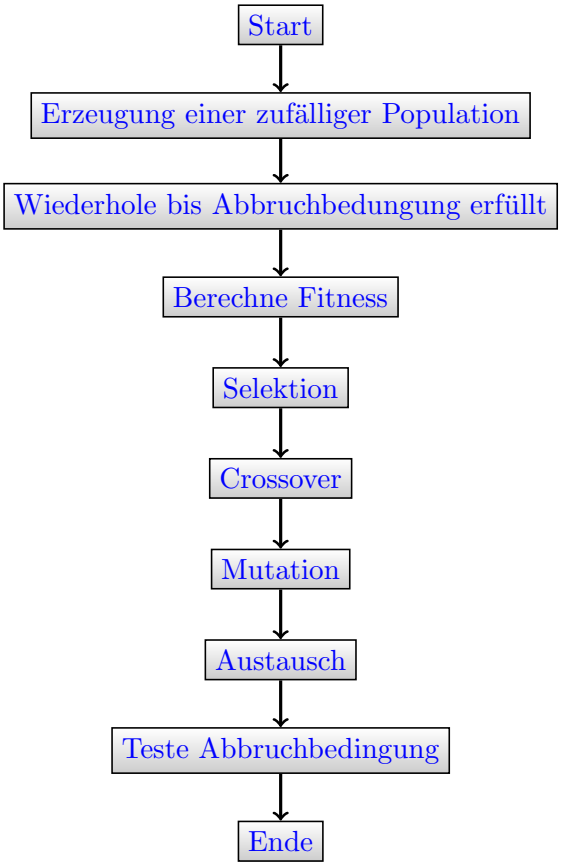
1.1 Problem des Handlungsreisenden

Das Problem des Handlungsreisenden ist ein kombinatorisches Optimierungsproblem der theoretischen Informatik. Dabei muss ein Handlungsreisender eine Menge von Städten besuchen. Er beginnt bei einer bestimmten Stadt und muss, nachdem jede andere Stadt besucht wurde, zu dieser zurückkehren. Das Optimierungsproblem besteht bei der Festlegung der Reihenfolge der zu besuchenden Städte, sodass die gesamte Distanz der Reise minimal ist. Das Problem ist als NP-vollständig klassifiziert.

1.2 Genetischer Algorithmus

Evolutionäre Algorithmen sind eine Klasse von stochastischen, heuristischen Optimierungsverfahren. Der Name lässt sich von der Evolution natürlicher Lebewesen ableiten. Ziel von genetischen Algorithmen ist es optimierte Lösungen zu Aufgabenstellungen zu finden, bei denen ein Auffinden einer akzeptablen Lösung aus Gründen der kombinatorischen Komplexität misslingt. So gibt es beim Problem des Handlungsreisenden mit 10 Orten z.B. bereits $10! = 3628800$ Lösungen. Kern eines genetischen Ansatzes ist die Veränderung von Mengen an Problemlösungen, sodass gute Lösungen mit einer großen Wahrscheinlichkeit und schlechte Lösungen mit geringen Wahrscheinlichkeit erhalten bleiben. Das Zusammenführen von Teilen guter Lösungen kann noch bessere Ergebnisse liefern. So kann verhindert werden, dass ein Algorithmus zur Optimierung an einem lokalen Optimum “hängenbleibt“.

1.2.1 Allgemeiner Ablauf



1.3 Chromosomen

Eine Rundreise des *Handlungsreisenden* wird duch ein Chromosom repräsentiert. Er bereist hierzu alle Städte und endet in der Stadt, in der begonnen hat. Die Population besteht aus der Anzahl solcher Rundreisen, die durch Vektoren modelliert werden.

Eine mögliche Rundreise wäre also: $\left\{ \begin{array}{c} 3 \\ 5 \\ 1 \\ 2 \\ 4 \\ 3 \end{array} \right\}$

1.4 Fitness

Eine Berechnung der Fitness in Abhängigkeit von der Gesamtdistanz einer Rundreise bietet sich für das *Problem des Handlungsreisenden* an. Allerding soll der Fitness-Wert proportional zur Eignung (antiproportional zu den Kosten) eines Chromosoms sein. Für die Kosten wird die Gesamtdistanz eines Chromosoms, für die Fitness der inverse Kostenwert verwendet.

1.5 Selektion

Die Auswahl eines Chromosomenpaars aus der Population soll wahrscheinlicher werden, wenn der Fitness-Wert steigt. Dies kann mittels einer Summe über alle Fitness-Werte gleichverteilte Zufallszahl realisiert werden. Der Wertebereich der Zufallszahl wird in Abschnitte unterteilt, deren Größe genau den Fitnesswerten der einzelnen Chromosomen entsprechen. Der Abschnitt, in den die Zufallszahl fällt, bestimmt die Selektion des zugehörigen Chromosoms.

1.6 Kreuzung

Ein zufälliger Kreuzungspunkt innerhalb des Vektors eines selektierten Chromosoms wird bestimmt. Der neue Vektor ist bis zum Kreuzpunkt identisch mit dem ursprünglichen. Der restliche Teil des neuen Vektors wird wie folgt bestimmt. Gehe den zweiten ausgewählten Vektor vom Anfang bis zum Ende durch und füge die erste Zahl, die noch nicht im neu zu bildenden Chromosoms steht, an die nächste freie Stelle ein. An der letzten Stelle des neuen Chromosomes muss die gleiche Zahl stehen wie an seiner ersten Stelle. Dieser Prozess wird für beide ausgewählten Chromosomen durchgeführt.

1.7 Mutation

Mit einer kleinen Wahrscheinlichkeit findet in der Kreuzung eine Mutation statt. Diese wird wie folgt implementiert. Ein zufälliges Paar von Indizes mit Werten zwischen 1 und der Anzahl der Städte wird erzeugt. Die Werte, die den Indizes entsprechenm werden im Chromosom getauscht. Dabei darf die Eigenschaft, dass Start- und Zielwert identisch sind, nicht verletzt werden.

1.8 Austausch

In jeder Iteration werden in der Population die beiden schlechtesten Chromosomen, d.h. diejenigen mit den geringsten Fitness-Werten, durch die beiden

neuen Chromosomen ersetzt, falls letztere bessere Fitness-Werte haben. Es wird gewährleistet, dass in der Population alle Chromosomen verschieden sind.

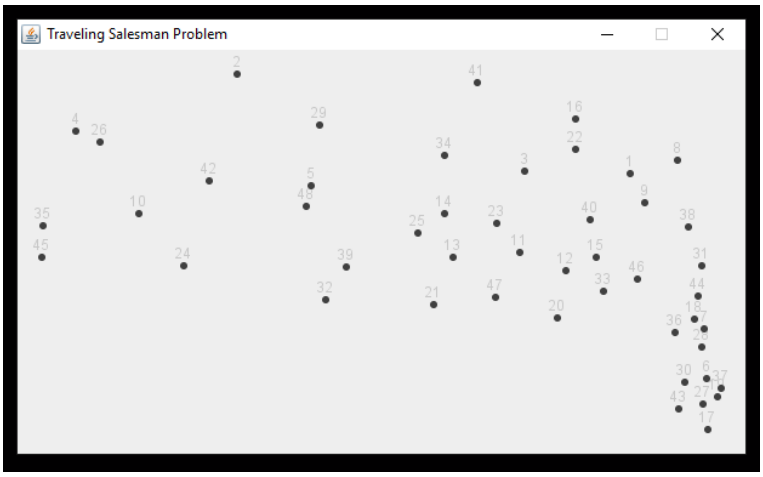
1.9 Realisierung mit Java

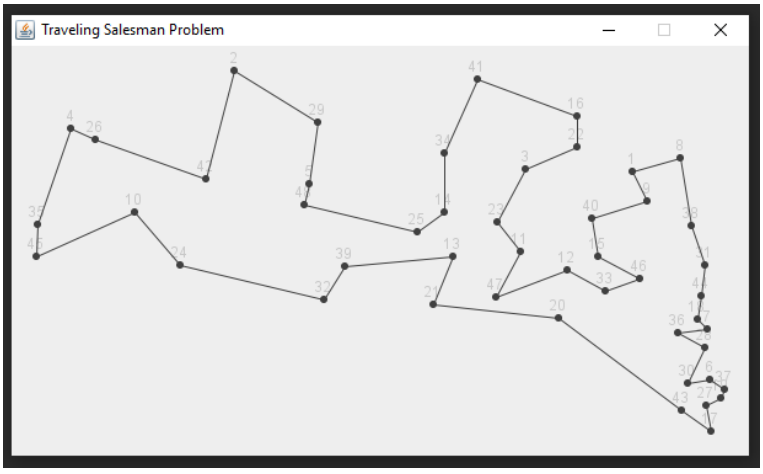
1.9.1 Eigenschaften

```
----------Genetic Algorithm Properties----------
Number of Cities:   48
Population Size:    500
Max. Generation:    500
k Value:            3
Elitism Value:      1
Force Uniqueness:   false
Local Search Rate:  0.0
Crossover Type:     UNIFORM_ORDER
Crossover Rate:     90.0%
Mutation Type:      INSERTION
Mutation Rate:      4.0%
```

```
-----------Genetic Algorithm Results------------
Average Distance of First Generation:  158045
Average Distance of Last Generation:   37931
Best Distance of First Generation:     126405
Best Distance of Last Generation:      36643
Area Under Average Distance:           33718137
Area Under Average Distance:           26872439
```

1.9.2 Visualisierung





2 Lernen

Dieses Kapitel beschreibt ein unüberwachtes Lernverfahren der Neuroinformatik: Die *Selbstorganisierenden Karten*, *Kohonen-Karten*, oder *Kohonen Feature Maps*. Also maschinelles Lernen ohne oder vorher bekannte Ergebniswerte.

2.1 Motivation

Das Funktionsprinzip von *SOM*s (*Self-organizing maps*) beruht auf der Erkenntnis, dass viele Strukturen im Gehirn eine lineare oder planare Topologie aufweisen. Eingehende Signale, wie z.B. visuelle Reize, sind jedoch multidimensional. Abbildung 1 zeigt eine Skizze der topologischen Ansicht des sensorischen Kotex, der die sensorischen Eingaben des Körpers auswertet. Es entsteht eine Klassifizierung, da ähnliche Ausgaben in direkter Umgebung angesiedelt werden. Beispielsweise sind Signale an die Finger in unmittelbarer Nachbarschaft zueinander. Solch eine Nachbarschaftsbeziehung zwischen Neuronen ist ein grundlegendes Merkmal von *SOM*s

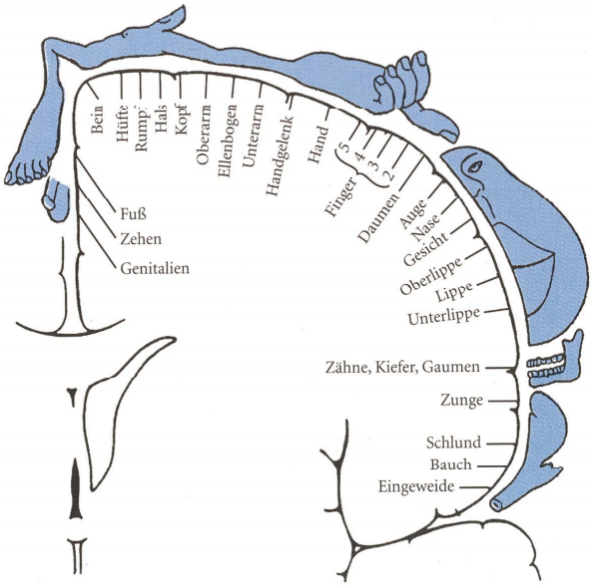


Abbildung 1: Topologische Abbildung des sensorischen Kortex

Eine topologieerhaltende Abbildung auf weniger Dimensionen (siehe Kapitel **Motivation**), hier eine Fläche, wird als Karte bezeichnet. Diese Karten im Gehirn kommen bei allen Säugetieren vor und sind dynamisch. Dies wird an folgendem Beispiel deutlich. Beim Verlust von Körperteilen wird der Platz des entsprechenden Feldes mit der Zeit von anderen Feldern eingenommen (da die eingehenden Signale wegfallen, Abbildung 2).

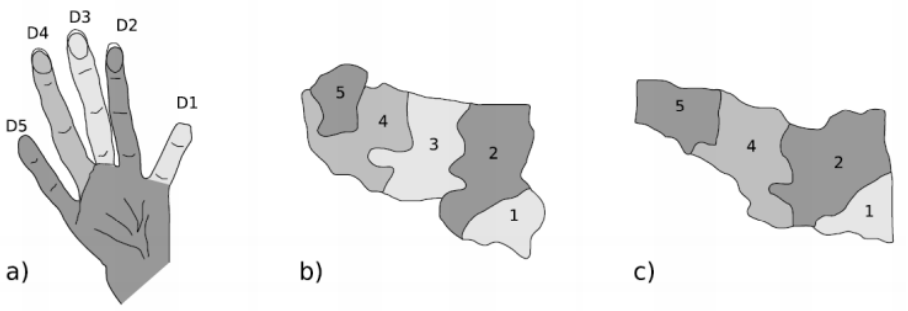


Abbildung 2: a) Schema einer Affenhand b) Bereich des Kortex der Hand c) 2 Monate nach Amputation von Finger D3

2.2 Aufbau

*SOM*s verwenden ein sehr einfaches Neuronenmodell ohne Aktivierungsfunktion (Abbildung 3)

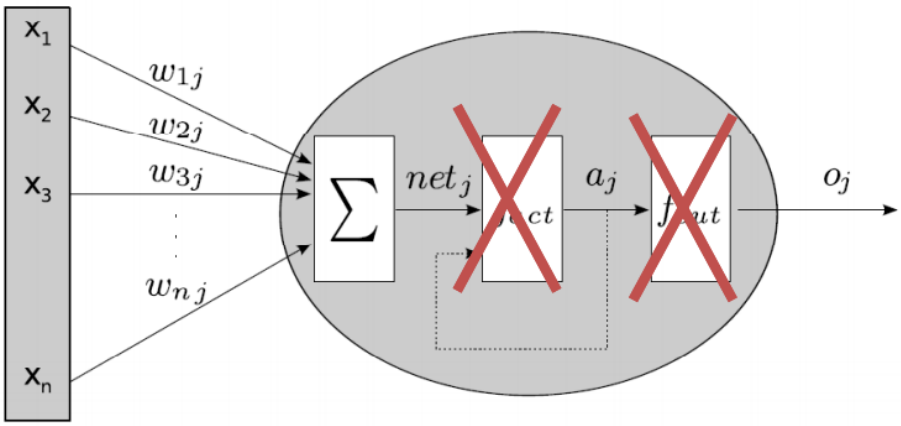


Abbildung 3: Neuron einer *SOM*

2.3 Netz

Organisiert man Neuronen als Eingabe- und Ausgabeneuronen in jeweils einer Schicht, entsteht ein neuronales Netz, in dem die Eingabeneuronen über die Gewichtsvektoren komplett mit den Ausgabeneuronen verbunden sind (Abbildung 4).

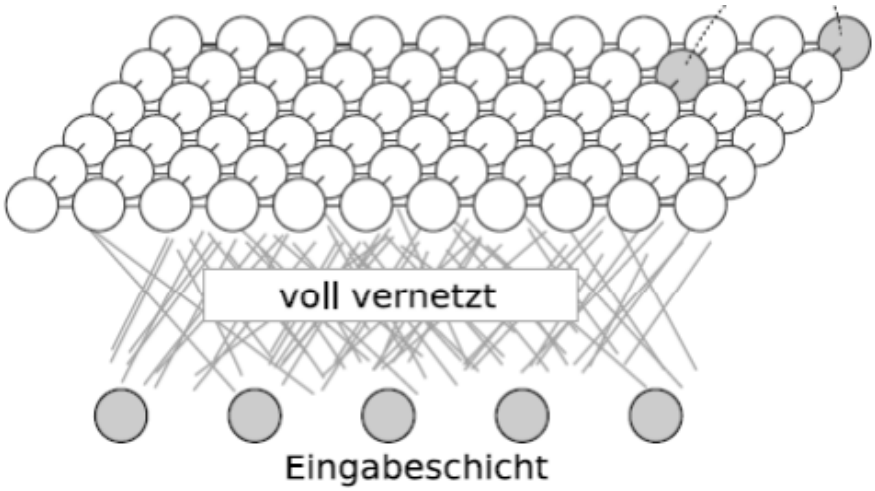


Abbildung 4: *Kohonen*-Netz

2.4 Lernverfahren

*SOM*s lernen durch Auswahl eines Neurons anhand deren „Eignung“. Die Eignung wird über die euklidische Norm berechnet. Hierbei wird der Abstand zwischen Eingabevektor und Gewichtsvektor bestimmt. Besitzen Neuronen die gleiche euklidische Norm, wird über das Zufallsprinzip entschieden, welches Neuron ausgewählt wird. Das Besondere an *SOM*s ist, dass nicht nur der Gewichtsvektor des ausgewählten Neurons (Siegerneuron) verändert wird, sondern auch die der Neuronen in der Umgebung des Siegerneurons.

2.5 Distanzfunktion

Anstelle der euklidischen Norm werden in der Praxis auch andere bewährte Distanzfunktionen verwendet. Einige Beispiele sind die gaussche Glockenfunktion, die Zylinderfunktion und der „Mexican-Hat“ (Abbildung 5).

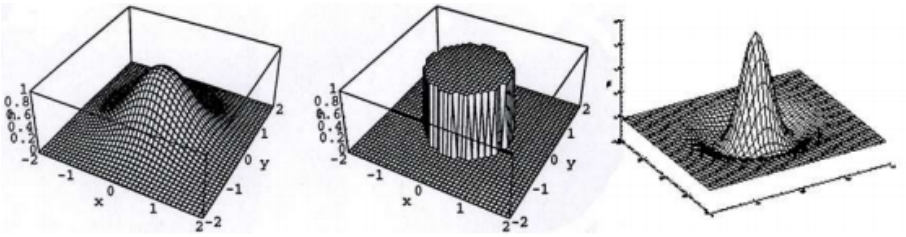


Abbildung 5: Gaussche Glockenfunktion, Zylinderfunktion und Mexican-Hat

2.6 Siegerneuron anpassen

$$W_c = W_c + \Delta W_c \text{ und } \Delta W_c = \eta * (x - W_c)$$

Mit c als Siegerneuron, W_c als Gewichtungsvektor und Eingabevektor x.
Es gilt $0 < \eta < 1$.

2.7 Nachbarneuronen des Siegerneurons anpassen

Nachbarn des Siegerneurons lernen umso stärker, je näher sie sich am Siegerneuron befinden:

$$\Delta W_j = \eta * h_{cj} * (x - W_j)$$

Mit h_{cj} als Nachbarschaftsfunktion (Wie stark lernt das Neuron j mit bei Siegerneuron c)
Als Funktion des Abstandes z der Neuronen wird in der Praxis oft eine Zylinderfunktion gewählt:

$$Zylinder : h_{cj}(z) = \begin{cases} 1 & falls\ z < d \\ 0 & sonst. \end{cases}$$

2.8 Visualisierung

Beispiel ‚Zoo'

animal	hair	feathers	eggs	milk	airborne	...
Erdferkel	true	false	false	true	false	false
Antilope	true	false	false	true	false	false
Seebarsch	false	false	true	false	false	true
Bär	true	false	false	true	false	false
Eber	true	false	false	true	false	false
Büffel	true	false	false	true	false	false
Kalb	true	false	false	true	false	false
Karpfen	false	false	true	false	false	true
Wels	false	false	true	false	false	true
Meerschweinchen	true	false	false	true	false	false
Gepard	true	false	false	true	false	false
Huhn	false	true	true	false	true	false
Döbel	false	false	true	false	false	true
Muschel	false	false	true	false	false	false
Krabbe	false	false	true	false	false	true
...						

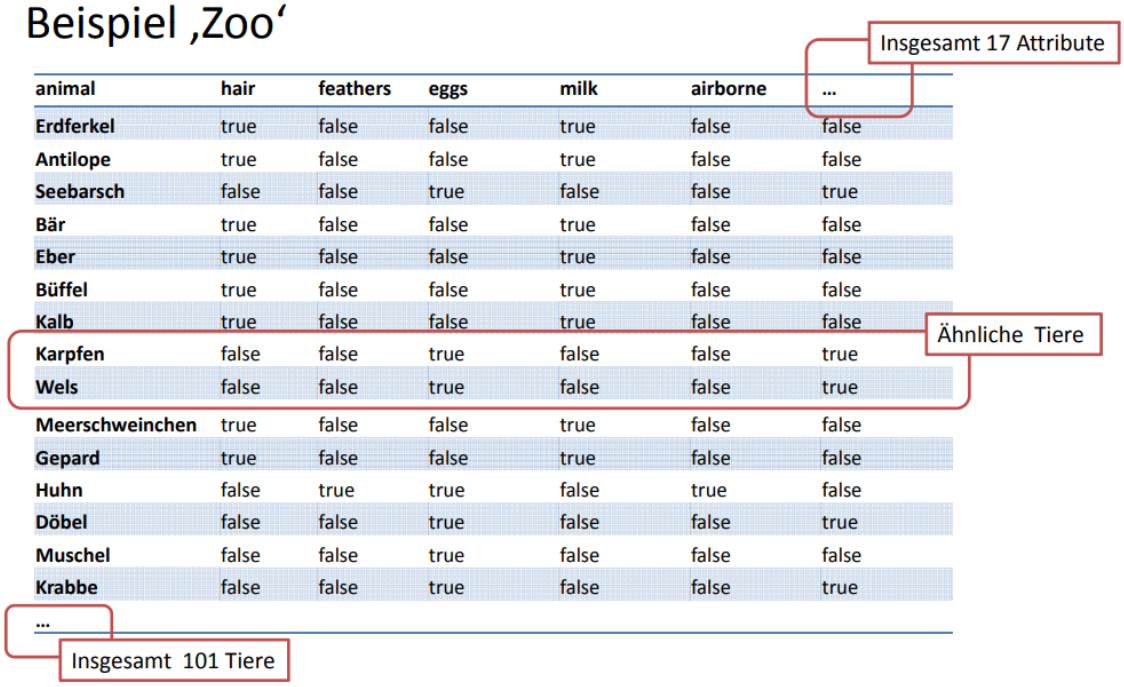


Abbildung 6: Beispiel „Zoo"

Reduktion von mehreren Dimension auf 2 (Karte). Sind ähnliche Tiere eines Zoos benachbart? (Abbildung 6+7)

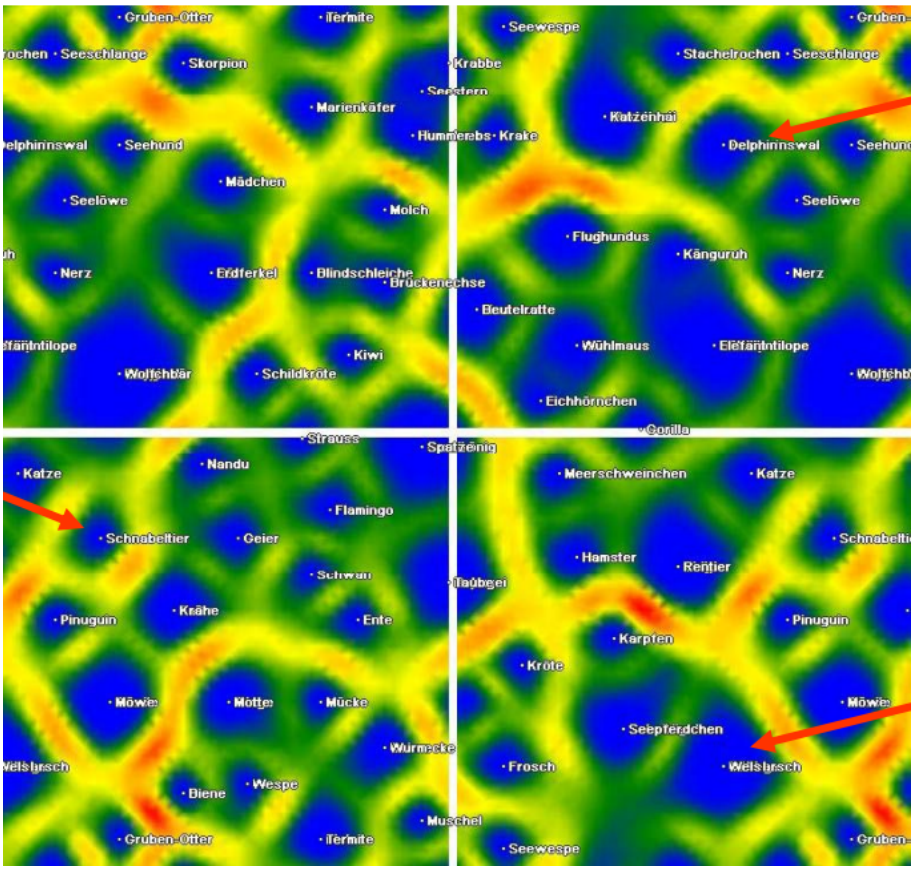


Abbildung 7: Beispiel „Zoo“

2.9 Anwendungsgebiete

- Dimensionsreduktion
- Optimierung
- Data Mining- Cluster
- Data Mining - Regeln
- Überwachenung und Anomaliedetektion
- Kontextkarten

3 Sequenzen

Die Verarbeitung von Sensordaten mobiler Endgeräte (z.B. Smartphones) wird in diesem Kapitel vorgestellt. Hierzu wird ein *Deep Learning* Ansatz gewählt.

3.1 Motivation

Die zunehmende Beliebtheit tragbarer Endgeräte der letzten Jahre gibt die Möglichkeit eine Vielzahl von physiologischen und funktionellen Daten kontinuierlich zu erfassen. Diese können Anwendungen für Sport, Gesundheit und Wohlbefinden zur Verfügung gestellt werden. Die Masse an Informationen erfordert effiziente Klassifizierungsmethoden und sinnvolle Analyse, wobei ein *Deep Learning* Ansatz eine vielversprechende Technik für die Datenanalyse im großen Stil ist. Die *Human Activity Recognition* (HAR) wertet zeiliche Reihendaten von Trägheitssensoren aus, um eine Aktion identifizieren zu können, die ausgeführt wird. So kann sowohl einen Ausbruch von Krankheiten (z.B. Parkinson), als auch die Wirksamkeit einer Behandlung erkannt werden.

3.2 Problem

Während *Deep Learning* bei Implementationen, die Hochleistungs-Rechencluster nutzen, bisher sehr erfolgreich war, ist der Einsatz auf mobilen Endgeräten durch geringe Ressourcen beschränkt.

3.3 Technik

Eine der größten Herausforderungen beim Entwerfen einer Klassifizierungsmethode für eine Zeitreihenanalyse ist die Auswahl von Merkmalen der Klassen. Oft kann durch den Einsatz von *Deep Believe Networks* (DBN), *Restricted Boltzman Machines* (RBM) oder *Convolutional Neural Networks* (CNN) eine Menge von *Features* (Markmale) der Rohdaten herausgearbeitet werden. Um eine vollständige Hierarchie der Features herauszufinden, die die Rohdaten sinnvoll klassifizieren kann, wird bei *HAR* jedoch ein *Deep Learnung* Ansatz mit komplexen Designs und vielen Schichten benötigt, um Änderungen in den verschiedenen Sensoren verarbeiten zu können.

3.4 „Shallow Deep Learning“ Methode

Im folgenden werden die einzelnen Schritte der vorgeschlagenen Methode gezeigt. Diese setzt sich aus *Deep Learning* (Prozess A) und *Shallow Learning* (Prozess B) zusammen.

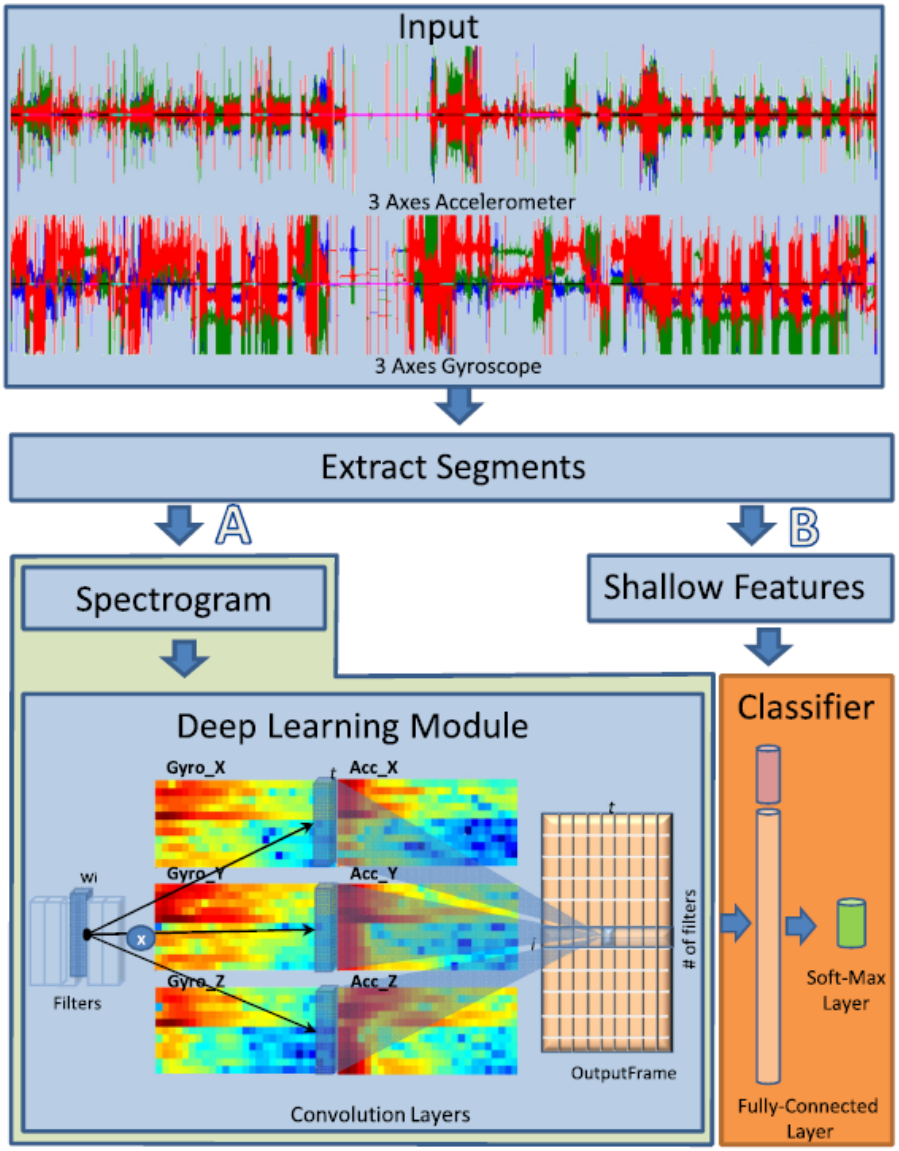


Abbildung 8: Schematischer Ablauf der Methode

3.4.1 Input

Um eingehende Daten (Input) zu generieren, wird auf verbaute Trägheitssensoren wie Beschleunigungsmesser und Gyroskope zugegriffen. Das Nutzen von Zeitreihendaten von anderen Sensoren, wie Elektrokardiographie (EKG) oder Elektromyographie (EMG), sind ebenfalls möglich.

3.4.2 Segmente extrahieren

Nachdem die Rohdaten eingesammelt werden, können n Segmente extrahiert werden, wobei n von der Art der Applikation abhängt. Die Länge der Segmente wird bei *HAR* auf 4 bis 10 Sekunden (*sliding window*) gesetzt, um die Genauigkeit der Erkennung der ausgeführten Aktivität zu maximieren, ohne die Grenzen zwischen verschiedenen Aktivitäten zu verschleiern.

3.4.3 Spektogramm und Deep Learning Modul

Um nun *Features* zu extrahieren, werden die Segmente als Spektogramm-Repräsentation an ein *Deep Learning* Modul weitergereicht, das darauf ausgelegt ist Intensitätsunterschiede in den Trägheitspunkten des Spektogramms zu interpretieren. Die Zeit- und Abtastrateninvarianz, die Amplituden und die Frequenz kann zur Untersuchung der Spektogramme genutzt werden (z.B. durch *Fourier*-Transformation).
Es ergeben sich beispielsweise folgende *Features*: (Abbildung 9)

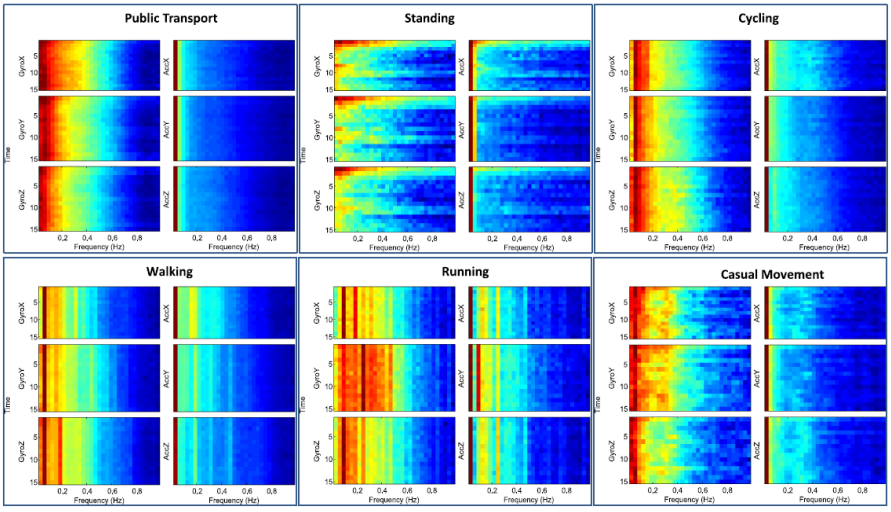


Abbildung 9: Spektogramme aus verschiedenen Aktivitäten

3.4.4 Shallow Features

Aus verschiedenen Berechnungen („flache“ Merkmale), die auf die Spektogramme angewendet werden (z.B. Quadratischer Mittelwert, Standartabweichung, usw.), kann nun eine Vektorrepräsentation jedes Segments erstellt werden.

3.4.5 Klassifizierung

Sobald sowohl flache (*shallow learned*), als auch tiefe (*deep learned*) Eigenschaften berechnet wurden, können diese zu einem eindeutigen Vektor zusammengeführt und klassifiziert werden.

3.4.6 Training

Flache und tiefe Features werden zusammen in einem einheitlichen, neuronalem Netz trainiert. In jeder Schicht des neuronalen Netzes werden Fehler (Errors) zwischen den Soll- und Istwerten in einer *Backwards Propagation* Routine genutzt, um die Gewichtungen in den *Hidden-Layers* anzupassen.

3.5 Evaluation (Vorschlag)

Das vorgeschlagene *Shallow Deep Learning*-Verfahren könnte nun mit verschiedenen Datensätzen (Abbildung 10) analysiert und evaluiert werden.

Dataset	Description	# of Classes	Subjects	Samples	Sampling Rate
ActiveMiles	Daily activities collected by smartphone in uncontrolled environments	7	10	4,390,726	50 – 200 Hz
WISDM v1.1	Daily activities collected by smartphone in a laboratory	6	29	1,098,207	20 Hz
WISDM v2.0	Daily activities collected by smartphone in uncontrolled environments	6	563	2,980,765	20 Hz
Daphnet FoG	Freezing of gait episodes in Parkinson's patients	2	10	1,917,887	64 Hz
Skoda	Manipulative gestures performed in a car maintenance scenario	10	1	$\sim 701,440$	98 Hz

Abbildung 10: Datensätze für eine Evaluierung

4 Ethik

5 Quellen