Mustererkennung - Übung7

Semjon Kerner, Philip Schmidt, Samuel Gfrörer

2016-12-19

1 Neuronale Netze ("backpropagation learning")

1.1 Ausführen des Programms

Das Programm kann folgendermaßen ausgeführt werden:

```
$ ./scripts/run.sh
```

1.2 Implementierung

Der gesamte Code sowie die ausführbare Datei befindet sich im beigefügten Archiv. Der Code wird als Github-Repository verwaltet (siehe https://github.com/EsGeh/pattern-recognition). Um das Programm selbst zu installieren und zu kompilieren, siehe unten.

1.2.1 Ordnerstruktur

```
.
|-- app
|   `-- Main.hs
|-- resource
|   |-- ...
|-- src
|   |-- PatternRecogn
|   |   `-- NeuronalNetworks.hs
|   `-- ...
```

Das Programm teilt sich auf in eine ausführbare Datei (siehe Verzeichnis "./app") und eine Bibliothek (siehe Verzeichnis "./src"). Für diese Übung relevanter Quellcode:

- "./app/Main.hs": hier befindet sich der Code zum Testen des Klassifizierungsalgorithmus anhand verschiedener Daten
- "./src/PatternRecogn/NeuronalNetworks.hs": Hier befindet sich die eigentliche Funktionalität des Klassifizierungsalgorithmus

1. Eingabe- und Ausgabedateien

Im Ordner "./resource" befinden sich die Trainingsdatensätze und der Testdatensatz. Die Ausgabe erfolgt über die Standardausgabe.

1.2.2 Die Funktionalität des Programms

Das Programm trainiert nacheinander künstliche neuronale Netzwerke mittels "Backpropagation Learning" zur Erkennung folgender Funktionen:

- logisches "UND"
- logisches "ODER"
- logisches "XOR"
- Klassifizierung der Trainingsdatensätze (Erkennung von Ziffern 3,5,7,8) in "./resource/train.*" und klassifiziert danach die Testdaten in "./resource/zip.test"

Dabei wird die Qualität der Klassifizierung über mehrere Iterationen ausgegeben. Im Falle der logischen Verknüpfungen sind Trainingsdaten und Testdaten identisch. Im Fall der Ziffern wird die Qualität sowohl für die Trainingsdaten als auch die Testdaten ausgegeben.

1.2.3 Ergebnis

Aus der Ausgabe des Programms (s.u.) wird Folgendes sichtbar:

- Die logischen Operationen "UND" und "ODER" werden sehr schnell erfolgreich gelern
- Die Testdaten werden mittels eines einfachen Netzes relativ schnell bis zu einer Erfolgsrate von ~0.98 (Trainingsdaten) gelernt. Der erlernte Klassifikator klassifiziert die Testdatenmenge sehr erfolgreich mit einer Erfolgsrate von ~0.9

Allgemeine Beobachtungen:

- Beim Lernen kann die Erfolgsrate durchaus vorübergehend wieder fallen
- Beim Lernen mit mehr als 1 Zwischenschicht verlangsamt sich der Lern-Prozess drastisch

1.2.3.1 Ausgabe des Programms

Erklärung der Ausgabe:

- "network dimensions [i,j,...]" bezeichnet ein Netzwerk, das außer der Eingabeschickt weitere Schichten mit der Knotenanzahl i,j,... aufweist.
- Ein Lernvorgang bricht ab (z.B. "stopped after ... iterations. Reason: ... ", wenn
  - nach einer maximalen Anzahl von Iterationen
  - falls die Erfolgsrate 1 beträgt
  - falls kein nennenswerter Fortschritt mehr gemacht wird

Dies ist die Ausgabe des Programms:

```
$ ./scripts/run.sh
-----------------------------------------
testing with operator "and"...
network dimensions: [2]
stopped after 130 iterations.
                Reason: quality >= 1.0
quality of classifying training data: 1.0
-----------------------------------------
testing with operator "or"...
network dimensions: [2]
stopped after 151 iterations.
                Reason: quality >= 1.0
quality of classifying training data: 1.0
-----------------------------------------
testing with operator "xor"...
network dimensions: [3,2]
iteration: 0
quality of classifying training data: 0.5
iteration: 100
quality of classifying training data: 0.5
iteration: 200
quality of classifying training data: 0.5
iteration: 300
quality of classifying training data: 0.5
iteration: 400
quality of classifying training data: 0.5
iteration: 500
quality of classifying training data: 0.75
iteration: 600
quality of classifying training data: 0.75
...
iteration: 9900
quality of classifying training data: 0.75
stopped.
                Reason: iterations >=10000
quality of classifying training data: 0.75
```

```
----------------------------------------
testing to classify test data (from file)...
network dimensions: [10]
iteration: 0
quality of classifying training data: 0.0
quality of classifying test data: 0.0
iteration: 100
quality of classifying training data: 0.9362765514369012
quality of classifying test data: 0.8575899843505478
iteration: 200
quality of classifying training data: 0.9712619741774261
quality of classifying test data: 0.8967136150234741
iteration: 300
quality of classifying training data: 0.973344439816743
quality of classifying test data: 0.8998435054773083
iteration: 400
quality of classifying training data: 0.9800083298625573
quality of classifying test data: 0.9061032863849765
iteration: 500
quality of classifying training data: 0.9762598917117867
quality of classifying test data: 0.9045383411580594
iteration: 600
quality of classifying training data: 0.9816743023740109
quality of classifying test data: 0.9092331768388107
iteration: 700
quality of classifying training data: 0.9816743023740109
quality of classifying test data: 0.9076682316118936
iteration: 800
quality of classifying training data: 0.9816743023740109
quality of classifying test data: 0.9076682316118936
iteration: 900
quality of classifying training data: 0.9816743023740109
quality of classifying test data: 0.9076682316118936
stopped.
               Reason: iterations >=1000
quality of classifying training data: 0.9816743023740109
quality of classifying test data: 0.9076682316118936
----------------------------------------
testing to classify test data (from file)...
network dimensions: [2,10]
iteration: 0
quality of classifying training data: 0.0
quality of classifying test data: 0.0
iteration: 100
quality of classifying training data: 0.27405247813411077
quality of classifying test data: 0.2597809076682316
```

```
iteration: 200
quality of classifying training data: 0.4868804664723032
quality of classifying test data: 0.4616588419405321
iteration: 300
quality of classifying training data: 0.48854643898375677
quality of classifying test data: 0.4460093896713615
iteration: 400
quality of classifying training data: 0.4947938359017076
quality of classifying test data: 0.45539906103286387
iteration: 500
quality of classifying training data: 0.4947938359017076
quality of classifying test data: 0.4538341158059468
iteration: 600
quality of classifying training data: 0.48771345272802996
quality of classifying test data: 0.4507042253521127
iteration: 700
quality of classifying training data: 0.49604331528529777
quality of classifying test data: 0.4522691705790297
iteration: 800
quality of classifying training data: 0.49604331528529777
quality of classifying test data: 0.4538341158059468
iteration: 900
quality of classifying training data: 0.49604331528529777
quality of classifying test data: 0.4538341158059468
stopped after 985 iterations.
               Reason: progress < 1.0e-2
quality of classifying training data: 0.49604331528529777
quality of classifying test data: 0.4538341158059468
```

1.3 Kompilieren des Programms

1.3.1 Abhängigkeiten

- git (siehe https://git-scm.com/)
- stack (siehe https://docs.haskellstack.org/)

1.3.2 Kompilieren

```
$ git clone https://github.com/EsGeh/pattern-recognition
$ git checkout exercise7-release
$ stack setup
$ stack build
```

1.3.3 Ausführen mittels Stack

```
$ stack exec patternRecogn-exe
```