Backpropagation: Lernen mit mehrschichtigen künstliches neuronalen Netzen

Richard Mrosk, Maximilian-Felix Helm

Hochschule Zittau / Görlitz

20. November 2023

Backpropagation: Lernen mit mehrschichtigen künstliches neuronalen Netzen

Richard Mrosk, Maximilian-Felix Helm

Hochschule Zittau / Görlitz

20. November 2023

Abstract

Bis jetzt kennen wir ja Neuronale-Netze mit mehreren Neuronen aber nur einen Layer.
Wenn man nun mehrere Layer nutzen möchte, wie sollte die Korrektur der Gewichte durch die Schichten gereicht werden?

Abstract

Bis jetzt kennen wir ja Neuronale-Netze mit mehreren Neuronen aber nur einen Layer.
Wenn man nun mehrere Layer nutzen möchte, wie sollte die Korrektur der Gewichte durch die Schichten gereicht werden?

fist note

second note

Einleitung

Steckbrief

- gehört zur Gruppe der überwachten Lernverfahren
- ist eine Verallgemeinerung der Delta-Regel für mehrschichtige Netze
- ist ein spezialfall des Gradientenverfahrens in der Optimierung

Einleitung

Steckbrief
- gehört zur Gruppe der überwachten Lernverfahren
- ist eine Verallgemeinerung der Delta-Regel für mehrschichtige Netze
- ist ein spezialfall des Gradientenverfahrens in der Optimierung

Einleitung

Wieso sollte ich das nutzen?

- effizient
- Universität
- mathematisch fundiert und gut erforscht
- viele Frameworks und Bibliotheken
- sehr gut skalierbarkeit
- Regularisierung und Optimierungstechniken

Einleitung

Wieso sollte ich das nutzen?
- effizient
- Universität
- mathematisch fundiert und gut erforscht
- viele Frameworks und Bibliotheken
- sehr gut skalierbarkeit
- Regularisierung und Optimierungstechniken

Rückblick Perzeptron

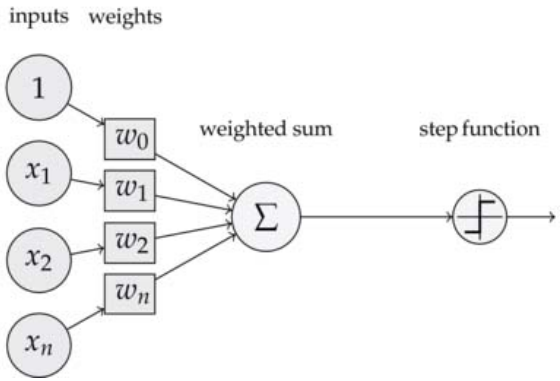
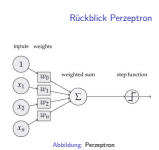


Abbildung: Perzeptron



Abbild einer biologischen Nervenzelle
Anhand der Inputs wird ein Output berechnet
Anpassungen der Gewichte geschehen direkt durch das Abgleichen des Outputs mit dem gewünschten Output
Optimum für den Trainingssatz wird durch das Senken und Erhöhen der Gewichte gesucht
Deltaregel wurde verwendet um zu einer schneleren Konvergenz zu gelangen

Rückblick Perzeptron

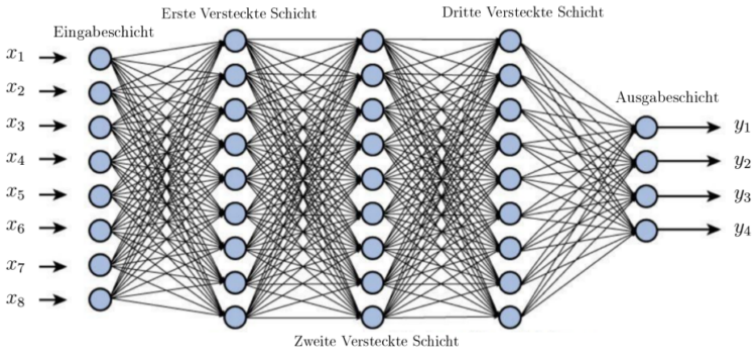
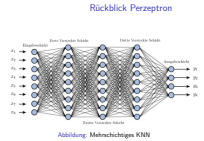


Abbildung: Mehrschichtiges KNN



Wie würdet ihr die Gewichte anpassen?

Video

.

[1]

Video

[1]

Funktionsweise

Vorwärtsdurchlauf

1. Berechne die Aktivierung für das aktuelle Neuron
2. Wiederhole Schritt 1 für alle Neuronen in der aktuellen Schicht
3. Wiederhole Schritt 1 & 2 für alle Schichten

$$n = \text{Anzahl der Inputs}$$
$$z^{(L)} = \sum_{i=0}^{n} Input_i * Gewicht_i$$
$$\text{Aktivierung} = Aktivierungsfunktion(z^{(L)} + bias)$$

Funktionsweise

Vorwärtsdurchlauf

1. Berechne die Aktivierung für das aktuelle Neuron
2. Wiederhole Schritt 1 für alle Neuronen in der aktuellen Schicht
3. Wiederhole Schritt 1 & 2 für alle Schichten

n = Anzahl der Inputs
$z^{(L)} = \sum_{i=0}^{n} Input_i * Gewicht_i$
Aktivierung = $Aktivierungsfunktion(z^{(L)} + bias)$

Funktionsweise

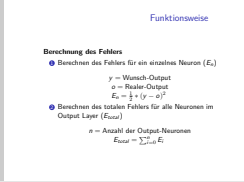
Berechnung des Fehlers

1. Berechnen des Fehlers für ein einzelnes Neuron (E_n)

$$y = \text{Wunsch-Output}$$
$$o = \text{Realer-Output}$$
$$E_n = \frac{1}{2} * (y - o)^2$$

2. Berechnen des totalen Fehlers für alle Neuronen im Output Layer (E_{total})

$$n = \text{Anzahl der Output-Neuronen}$$
$$E_{total} = \sum_{i=0}^{n} E_i$$



Funktionsweise

Rückwärtsdurchlauf

Partielle Ableitungen der Fehlerfunktion nach Gewichten w_i

$\frac{\partial E_{total}}{\partial w_i} = \frac{\partial E_{total}}{\partial out} * \frac{\partial out}{\partial s} * \frac{\partial s}{\partial w_i}$

$(s = X_i * w_i + \ldots + X_n * w_n + b)$

Partielle Ableitungen der einzelnen Bestandteile unter Verwendung der Sigmoid-Funktion als Aktivierungsfunktion:

$\frac{\partial E_{total}}{\partial out} = out - target$

$\frac{\partial out}{\partial s} = \frac{1}{(1+e^{-s})} * \left(1 - \frac{1}{1+e^{-s}}\right)$

$\frac{\partial s}{\partial w_i} = X_i$

[3]

Das alles gilt für ein Neuron in einen Layer
out := Funktion des Neurons (mit Inp. Vect X)
s := Aktivierung (Summe von X_i * w_i + b)

Funktionsweise

Diese Ableitungen nutzt man zum Errechnen des neuen Gewichts w_i:

$\epsilon =$ Lernrate
$w_i = w_i - \epsilon * \frac{\partial E_{total}}{\partial w_i}$

Funktionsweise

Diese Ableitungen nutzt man zum Errechnen des neuen Gewichts w_i:

$\epsilon =$ Lernrate
$w_i = w_i - \epsilon * \frac{\partial E_{total}}{\partial w_i}$

Weitere Verfahren

- Evolutionäre Algorithmen / Genetische Programmierung
- Hebbsche Lernregel
- Bestärkendes Lernen
- Konkurrenzfähige Lernmethoden

Weitere Verfahren

- Evolutionäre Algorithmen / Genetische Programmierung
- Hebbsche Lernregel
- Bestärkendes Lernen
- Konkurrenzfähige Lernmethoden

Erhöhung und Senkung der Verbindungen zwischen den Neuronen
Neuronen treten gegebeinander an

Die Bibliothek

- **Deeplearning4J** [2]
- verbreitet
- direkt in Java nutzbar
- schnell

Die Bibliothek
Deeplearning4J [2]
verbreitet
direkt in Java nutzbar
schnell
DL4J

1. ist eine Lib (-Sammlung) zur einfachen Konfiguration von Neuronalen Netzen und Co
2. 13.2 Stars on Github
3. -
4. wichtige Operationen + Arrays in C++ implementiert

Bau des Netzwerks

```
var builder = new NeuralNetConfiguration.Builder();
builder.weightInit(WeightInit.XAVIER)
  .activation(Activation.TANH)
  .updater(new Sgd(0.1))
  .weightDecay(1e-4);
// more configuration ...
builder.list()
  .layer(new DenseLayer.Builder().nIn(featureCount)
    .nOut(8).build())
  // opt. more layers
  .layer(new OutputLayer.Builder(
    LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
    .nIn(8).nOut(outputCount)
    .activation(Activation.SOFTMAX).build())
```


Bau des Netzwerks

var builder = new NeuralNetConfiguration.Builder();
builder.weightInit(WeightInit.XAVIER)
.activation(Activation.TANH)
.updater(new Sgd(0.1))
.weightDecay(1e-4);
// more configuration ...
builder.list()
.layer(new DenseLayer.Builder().nIn(featureCount)
.nOut(8).build())
// opt. more layers
.layer(new OutputLayer.Builder(
LossFunctions.LossFunction.NEGATIVELOGLIKELIHOOD)
.nIn(8).nOut(outputCount)
.activation(Activation.SOFTMAX).build())

- nur wirklich notwendige Konfigration erforderlich
- OutputLayer übernimmt übersetzung der Neuronen-Outputs in Wahrscheinlichkeiten für jeden möglichen Fall

Nutzung des Netzwerkes

```
public void train(DataSet trainingData) {
  for(int i = 0; i < 1000; i++) {
    model.fit(trainingData);
  }
}

public Type classify(INDArray inp) {
  final int[] result = model.predict(inp);
    return Type.getEntries().get(result[0]);
  }
}
```


Nutzung des Netzwerkes

public void train(DataSet trainingData) {
for(int i = 0; i < 1000; i++) {
model.fit(trainingData);
}
}

public Type classify(INDArray inp) {
final int[] result = model.predict(inp);
return Type.getEntries().get(result[0]);
}
}

- interface ähnlich zu bekannten Learner
- model: Instanz eines Netzes (durch vorherige Folie)
- DataSet: DL4J struct für 1..n FeatureVectors
- INDArray: DL4J struct für native arrays; hier für inp FetureVector

Import der FeatureVectors

- Daten von vorhergehender Übung übernommen
- DL4J erwartet *DataSet*
- Liste an FeatureVectors wird entpackt und in ein DataSet geschrieben
- an jeden FeatureVector wird die Ordinale des Konzeptes vorangestellt
- Feature-Werte werden normalisiert

Import der FeatureVectors

- Daten von vorhergehender Übung übernommen
- DL4J erwartet *DataSet*
- Liste an FeatureVectors wird entpackt und in ein DataSet geschrieben
- an jeden FeatureVector wird die Ordinale des Konzeptes vorangestellt
- Feature-Werte werden normalisiert

1. und so auch die Deserialisierung; cutting: cropped
2. DataSet kann 1..n FeatureVectors + Type enthalten (kümmert sich selber um zuordnung und splitting)
3. -
4. ... und dem DataSet mitgeteilt, dass der erste Wert das erwartete Konzept wiedergibt
5. DL4J hat dafür eingebaute Methoden

Evaluation

- analog zur vorherigen Übung
- Train + Test n-mal wiederholen und Durchschnitt mit 95% Konfidenzintervall berechnen
- BSP. Ausgabe: `Error-Rate for 95%: ErrRate[ rate=0.1307, confidence=7.1512E-5]`

Evaluation

- analog zur vorherigen Übung
- Train + Test n-mal wiederholen und Durchschnitt mit 95% Konfidenzintervall berechnen
- BSP. Ausgabe: Error-Rate for 95%: ErrRate[rate=0.1307, confidence=7.1512E-5]

1. 60% Trainingsdaten; Fehler = Mismatched + Unknown

Übung

Ziel:

- mit Deeplearning4J vertraut machen
- eine Backpropagation betrachten

Übung

Ziel:
- mit Deeplearning4J vertraut machen
- eine Backpropagation betrachten

Die Daten sind ähnlich zur 1. Aufgabe. Zur Einfachheit wurden aber die Konzepte reduziert auf *Vorfahrt* und *Other*

#DO download and setup
(Empfehlung: Gradle Cache nachher löschen, da das recht große dependencies sind)

1. durch das selbständige Konfigurieren eines Netzes
2. Programm gibt Werte aus

Aufgabe 1

Schauen Sie sich den Skeleton-Code and, um sich mit der Funktionsweise von Deeplearning4J vertraut zu machen (insbesondere *NeuralNetFactory* und *Learner*).

Bearbeiten Sie in *NeuralNetFactory* die TODOs mit "Task 1". Danach kann mit der Gradle-Task *run* die Evaluation gestartet werden.

Aufgabe 1

Schauen Sie sich den Skeleton-Code and, um sich mit der Funktionsweise von Deeplearning4J vertraut zu machen (insbesondere *NeuralNetFactory* und *Learner*).

Bearbeiten Sie in *NeuralNetFactory* die TODOs mit "Task 1". Danach kann mit der Gradle-Task *run* die Evaluation gestartet werden.

#NOTE sollte Zeit knapp werden, Eval überspringen (da es lange braucht)

Aufgabe 2

Stellen Sie im Lerner das if auf *false* um.
Führen Sie nun das Programm erneut aus und studieren die Ausgaben des Reporters.

Aufgabe 2

Stellen Sie im Lerner das if auf *false* um.
Führen Sie nun das Programm erneut aus und studieren die Ausgaben des Reporters.

Man sieht, ..., irgendwas

#NOTE wenn noch genug Zeit ist, gibt es im Lerner noch eine Aufgabe 3 (bezüglich des Batchings)

[1] 3Blue1Brown. *What is backpropagation really doing? | Chapter 3, Deep learning*. 3:07 - 12:28. 3. Nov. 2017. URL: https://iv.datura.network/watch?v=Ilg3gGewQ5U.

[2] *Deeplearning4J (Github)*. 14. Nov. 2023. URL: https://github.com/deeplearning4j/deeplearning4j.

[3] Ahmed Gad. *A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks*. 20. Nov. 2023. URL: https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide.

[1] 3Blue1Brown. *What is backpropagation really doing? | Chapter 3, Deep learning*. 3:07 - 12:28. 3. Nov. 2017. URL: https://iv.datura.network/watch?v=Ilg3gGewQ5U.
[2] *Deeplearning4J (Github)*. 14. Nov. 2023. URL: https://github.com/deeplearning4j/deeplearning4j.
[3] Ahmed Gad. *A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks*. 20. Nov. 2023. URL: https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide.