



Deep Learning

Einführung - Thema 2

Silas Hoffmann

11. April 2020

Fachhochschule Wedel

Deep Learning

2020-04-11



Deep Learning

Einführung - Thema 2

Silas Hoffmann
11. April 2020
Fachhochschule Wedel

Geschichtliche Entwicklung

McCulloch-Pitts-Neuron

Perceptron

Adeline

Backpropagation

Aktuelle Entwicklung

Convolutional Neural Network

Inhalt
Geschichtliche Entwicklung
McCulloch-Pitts-Neuron
Perceptron
Adeline
Backpropagation
Aktuelle Entwicklung
Convolutional Neural Network

2020-04-11

Deep Learning

└─ Geschichtliche Entwicklung

└─ McCulloch-Pitts-Neuron

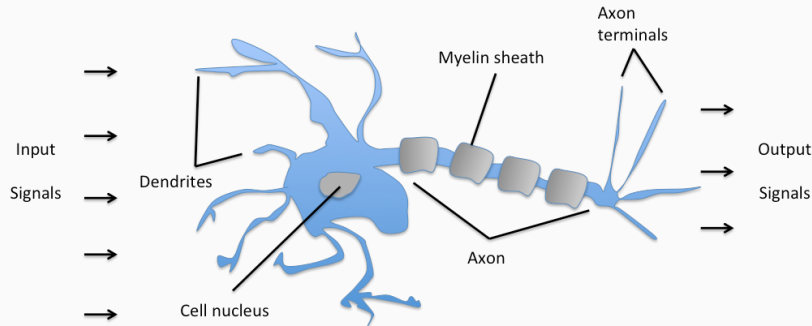
Geschichtliche Entwicklung

McCulloch-Pitts-Neuron

Geschichtliche Entwicklung

McCulloch-Pitts-Neuron

Zusammenhang - Biologisches Neuron



Schematic of a biological neuron.

2020-04-11

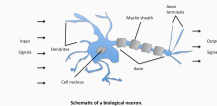
Deep Learning

└─ Geschichtliche Entwicklung

└─ McCulloch-Pitts-Neuron

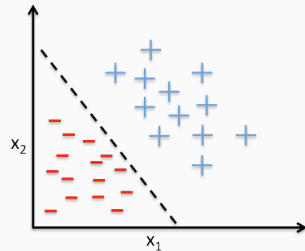
└─ Zusammenhang - Biologisches Neuron

Zusammenhang - Biologisches Neuron



1. *Dendriten*: Nehmen Infos auf
 - besitzen Rezeptoren und Signale anderer Dendriten aufzunehmen
2. *Signale*: bewirken elektrische Veränderungen
 - werden vom Zellkern (Soma) interpretiert / verarbeitet
 - Zellkern sammelt Infos, speichert diese im Axonhügel
3. Ursprung vom Axon / Neuriten
4. Wenn Signal stark genug: an Axon weitergeleitet
 - auch als *Aktionspotential bezeichnet*
 - Signal am Ende über Axonterminale per Neurotransmitter mit nächste Dendriten verbunden

- Modell soll Funktionalität des biologischen Neurons imitieren
- Klassifizierungsproblem als grundlegende Problemstellung
- Lineare Entscheidungsfunktion zur binären Klassifizierung verwendet



Example of a linear decision boundary for binary classification.

2020-04-11

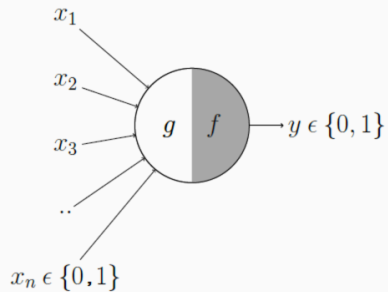
Deep Learning

- └ Geschichtliche Entwicklung
 - └ McCulloch-Pitts-Neuron
 - └ MP-Neuron

- Modell soll Funktionalität des biologischen Neurons imitieren
- Klassifizierungsproblem als grundlegende Problemstellung
- Lineare Entscheidungsfunktion zur binären Klassifizierung verwendet



1. 1943: Warren McCulloch & Walter Pitts
2. soll biologisches Neuron imitieren
3. Klassifizierungsproblem: anhand vom geg. Merkmalsvektor entscheiden ob Objekt X in Klasse K liegt
4. hier lediglich binäre Klassifikation
 - Unterscheidung nur zwischen zwei Klassen
 - Sonderfall dieses Modells: nur boolesche Eingabewerte
5. muss mittels linearer Entscheidungsfunktion definierbar sein



$$g(x_1, x_2, \dots, x_n) = g(x) = \sum_{i=1}^n x_i$$

$$f(g(x)) = \begin{cases} 1 & \text{if } g(x) \geq \theta \\ 0 & \text{if } g(x) < \theta \end{cases}$$

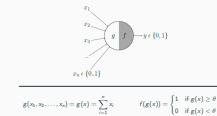
2020-04-11

Deep Learning

└ Geschichteliche Entwicklung

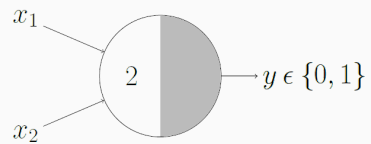
└ McCulloch-Pitts-Neuron

└ Aufbau und Funktionsweise



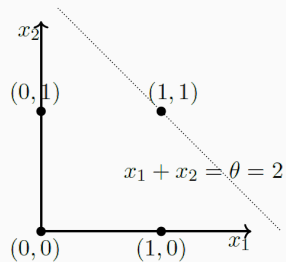
1. beliebig viele Eingabewerte
 - müssen boolescher Natur sein
2. Arbeitsschritte:
 - Alle Werte aufaddiert (Fkt. g)
 - Fkt. f prüft ob Schwellwert überschritten

Notation AND-Gatter



AND function

$$x_1 + x_2 = \sum_{i=1}^2 x_i \geq 2$$



2020-04-11

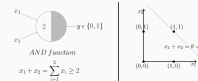
Deep Learning

└ Geschichtliche Entwicklung

└ McCulloch-Pitts-Neuron

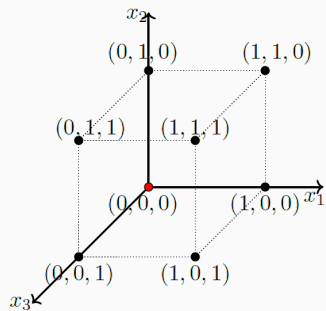
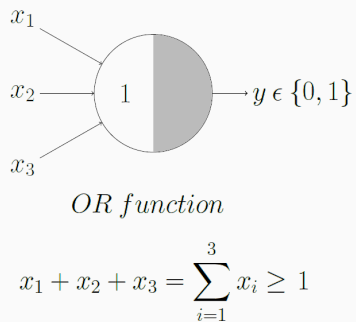
└ Notation AND-Gatter

Notation AND-Gatter



1. Anhand von Grafik erläutern

Notation OR-Gatter



Deep Learning

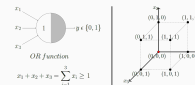
└ Geschichtliche Entwicklung

└ McCulloch-Pitts-Neuron

└ Notation OR-Gatter

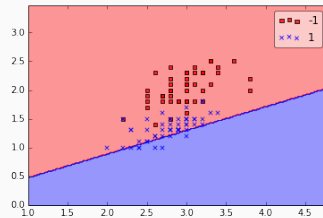
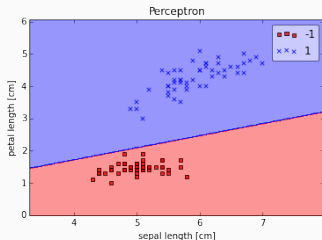
2020-04-11

Notation OR-Gatter



1. Anhand von Grafik erläutern, auch im 3d - Raum möglich

- Keine kontinuierlichen Eingabewerte (nur boolesche Werte)
- Schwelle muss manuell gesetzt werden, keine automatische Aktualisierung vorgesehen
- Keine Priorisierungsmöglichkeit der Eingabewerte möglich
- Funktionen müssen durch lineare Entscheidungsfunktion getrennt werden können



- Keine kontinuierlichen Eingabewerte (nur boolesche Werte)
- Schwelle muss manuell gesetzt werden, keine automatische Aktualisierung vorgesehen
- Keine Priorisierungsmöglichkeit der Eingabewerte möglich
- Funktionen müssen durch lineare Entscheidungsfunktion getrennt werden können



1. keine kontinuierlichen Eingabewerte
 - nur boolesche Werte
 - Schwierig für komplexe Anwendungen
 - siehe Bilderkennung - Farbwerte
2. Schwelle muss manuell gesetzt werden
 - Sprich kein Lernalgorithmus vorhanden
3. Keine Priorisierungsmöglichkeiten
 - siehe Gewichtete Eingaben
4. Funktionen durch lineare Entscheidungsfunktion getrennt
 - schwierig bei überlappenden Cluster
 - keine Polynome wie bei späteren Entwicklungen möglich
5. auch gedeckelte Fkt. wie XOR können nicht dargestellt werden
 - Schwelle muss genau getroffen werden

2020-04-11

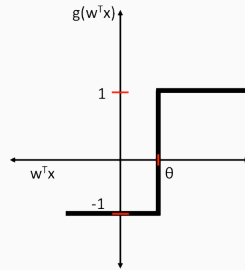
Deep Learning
└─ Geschichtliche Entwicklung
 └─ Perceptron

Geschichtliche Entwicklung
└─ Perceptron

Geschichtliche Entwicklung

Perceptron

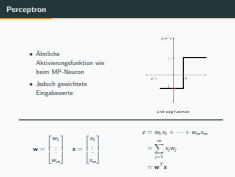
- Ähnliche Aktivierungsfunktion wie beim MP-Neuron
- Jedoch gewichtete Eingabewerte



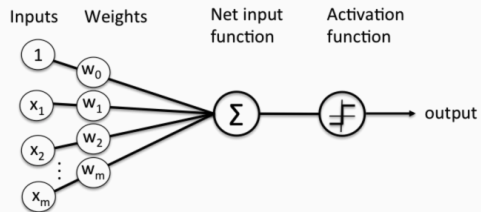
Unit step function.

$$\mathbf{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$$

$$\begin{aligned} z &= w_1 x_1 + \dots + w_m x_m \\ &= \sum_{j=1}^m x_j w_j \\ &= \mathbf{w}^T \mathbf{x} \end{aligned}$$



1. 1958: US-amerikanische Psychologe / Informatiker Frank Rosenblatt
2. älteste heutzutage noch genutzte NN
3. inspiriert vom Auge einer Fliege
 - Flugrichtung - Entscheidungen teils direkt im Auge getroffen
4. Weiterentwicklung der MP-Zelle
5. Eingabewerte mit Gewichten priorisiert
 - Auf Formel verweisen
6. Gleich bleibt jedoch die binäre Klassifikation
 - Verweis auf Unit step function
 - hier jedoch nicht Wahrheitswerte sondern -1 und 1



Schematic of Rosenblatt's perceptron.

$$g(z) = \begin{cases} 0 & \text{if } z \leq 0 \\ 1 & \text{if } z > 0 \end{cases}$$

$$\begin{aligned} z &= \mathbf{w_0x_0} + w_1x_1 + \dots + w_mx_m \\ &= \sum_{j=0}^m x_j w_j \\ &= \mathbf{w^T x} \end{aligned}$$

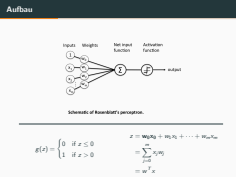
2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└ Perceptron

└ Aufbau



1. Grafik erläutern

2. Konvention:

- erleichtert später Notation der Lernregel
- Schwellwert auf andere Seite der z-Wert Gleichung ziehen

- Modell übernimmt selbst die Anpassung der Gewichte
- Test mittels einer Menge von gelabelten Trainingsdatensätzen

Grober Ablauf

- Initialisiere die Gewichte mit einem sehr kleinen Wert oder 0.
- Für jeden Datensatz der Menge von Trainingsdatensätzen:
 - Berechne den Ausgabewert des Systems
 - Gleiche die Gewichte an

- Modell übernimmt selbst die Anpassung der Gewichte
- Test mittels einer Menge von gelabelten Trainingsdatensätzen

Grober Ablauf

- Initialisiere die Gewichte mit einem sehr kleinen Wert oder 0.
- Für jeden Datensatz der Menge von Trainingsdatensätzen:
 - Berechne den Ausgabewert des Systems
 - Gleiche die Gewichte an

1. Rosenblatt erfindet *lernenden* Algorithmus
2. Auf Menge von Trainingsdatensätzen zurückgegriffen
 - Datensätze bestehen aus Ein- und erwarteten Ausgabewerten
 - in Literatur auch *gelabelte* Werte genannt
3. Lernalgorithmus - grobe Zusammenfassung
 - Gewichte mit kleinem Wert / 0 vorinitialisieren
 - Datensätze durchiterieren
 - Ausgabewert berechnen
 - Gewichte angleichen

Angleichung der Gewichte

- Gewichte komponentenweise angleichen: $w_j := w_j + \Delta w_j$
- Gewichtsänderung: $\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$

- Beispiel - Iteration mit zweidimensionalem Trainingsvektor:

$$\Delta w_0 = \eta (\text{target}^{(i)} - \text{output}^{(i)})$$

$$\Delta w_1 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_1^{(i)}$$

$$\Delta w_2 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_2^{(i)}$$

Angleichung der Gewichte

- Gewichte komponentenweise angleichen: $w_j := w_j + \Delta w_j$
- Gewichtsänderung: $\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$
- Beispiel - Iteration mit zweidimensionalem Trainingsvektor:

$$\Delta w_0 = \eta (\text{target}^{(i)} - \text{output}^{(i)})$$

$$\Delta w_1 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_1^{(i)}$$

$$\Delta w_2 = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_2^{(i)}$$

1. Erste Formel auf Slide beschreiben
 - Gewichte können zu Gewichtsvektor zusammengezogen werden
 - hier komponentenweise betrachtet
 - Delta (Dreieck) wird stets als Änderung verstanden
2. Exponent i hierbei jeweils als Index des Trainingsvektors in Menge
3. Lernalgorithmus arbeitet inkrementell
 - Lernrate (eta) bestimmt wie stark die Gewichte pro Durchlauf angeglichen werden
 - Differenz mit Lernrate und Eingabewert multipliziert
4. Iteration mit 2d - Eingabevektor
 - w_0 hierbei der Schwellwert selbst
 - Faktor x weggelassen da bereits gleich 1
 - Nutzung der beschriebenen Notation

Gewichtsänderung

$$\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$$

- Trainingsdatensatz richtig erkannt:

$$\Delta w_j = \eta((-1^{(i)}) - (-1^{(i)})) x_j^{(i)} = 0$$

$$\Delta w_j = \eta(1^{(i)} - 1^{(i)}) x_j^{(i)} = 0$$

- Trainingsdatensatz falsch erkannt:

$$\Delta w_j = \eta(1^{(i)} - (-1^{(i)})) x_j^{(i)} = \eta(2) x_j^{(i)}$$

$$\Delta w_j = \eta((-1^{(i)}) - 1^{(i)}) x_j^{(i)} = \eta(-2) x_j^{(i)}$$

2020-04-11

Deep Learning

Geschichtliche Entwicklung

Perceptron

Lernregel - Trainingsbeispiele

Gewichtsänderung

$$\Delta w_j = \eta (\text{target}^{(i)} - \text{output}^{(i)}) x_j^{(i)}$$

- Trainingsdatensatz richtig erkannt:

$$\Delta w_j = \eta((-1^{(i)}) - (-1^{(i)})) x_j^{(i)} = 0$$

$$\Delta w_j = \eta(1^{(i)} - 1^{(i)}) x_j^{(i)} = 0$$

- Trainingsdatensatz falsch erkannt:

$$\Delta w_j = \eta(1^{(i)} - (-1^{(i)})) x_j^{(i)} = \eta(2) x_j^{(i)}$$

$$\Delta w_j = \eta((-1^{(i)}) - 1^{(i)}) x_j^{(i)} = \eta(-2) x_j^{(i)}$$

1. Erinnerung: erst target dann output
2. Richtig erkannt
 - Generell Ausgabe 0, keine Änderung
 - Beide Falsch: -1
 - Beide Richtig: +1
3. Falsch erkannt
 - output zu klein
 - erwartetet +1 bekommen -1
 - Positiver (Differenz-)Faktor
 - output zu groß
 - erwartetet -1 bekommen +1
 - Negativer (Differenz-)Faktor

2020-04-11

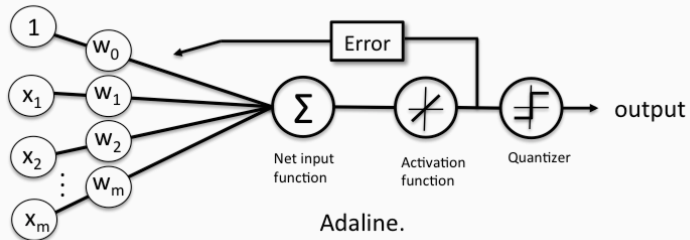
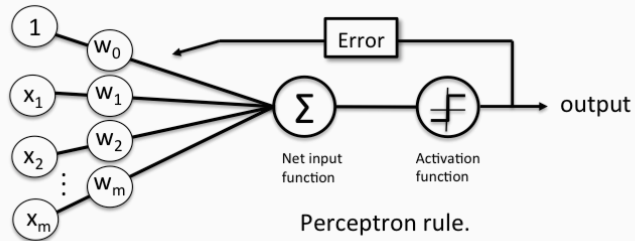
Deep Learning
└─ Geschichtliche Entwicklung
 └─ Adeline

Geschichtliche Entwicklung
Adeline

Geschichtliche Entwicklung

Adeline

ADaptive LINear Element



2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└ Adeline

└ ADaptive LINear Element

ADaptive LINear Element



1. 1959: Stanford Prof. Bernard Widrow & Elektroingenieur Marcian Edward Hoff
2. ADELIN: ADaptive LINear Element
3. Modell: Verzicht auf Einheitssprungfunktion bei Angleichung der Gewichte
 - Stattdessen lineare Aktivierungsfunktion
 - erstmal nur Identitätsfunktion verwendet
 - Entscheidungsfunktion für output weiterhin verwendet

Delta-Regel

- Lernalgorithmus durch Erfinder geprägt
- auch unter *Least-Mean-Square-Algorithmus* bekannt
- Wesentlicher Vorteil: Ableitbare Kostenfunktion

Notation

$$J(w) = \frac{1}{2} \sum_i (\text{target}^{(i)} - \text{output}^{(i)})^2 \quad \text{output}^{(i)} \in \mathbb{R}$$

2020-04-11

Deep Learning

└─ Geschichtliche Entwicklung

└─ Adeline

└─ Delta-Regel

- Lernalgorithmus durch Erfinder geprägt
- auch unter *Least-Mean-Square-Algorithmus* bekannt
- Wesentlicher Vorteil: Ableitbare Kostenfunktion

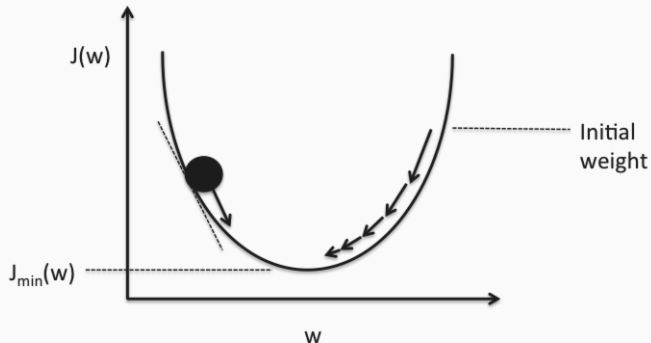
Notation

$$J(w) = \frac{1}{2} \sum_i (\text{target}^{(i)} - \text{output}^{(i)})^2 \quad \text{output}^{(i)} \in \mathbb{R}$$

1. Auch unter *Least-Mean-Square-Algorithmus* bzw. *Regressionsquadratsumme* bekannt
 - noch heute relevant
2. Funktion stellt Kostenfunktion dar
 - Fehler bei Kostenfunktion soll mithilfe der Lernregel minimiert werden
3. Vorteil dieses Ansatzes: Ableitbare Kostenfunktion
4. Formel erläutern:
 - Differenz quadriert um Vorzeichen zu verlieren
 - Faktor 1 / 2 verschieben um Ableitung einfacher zu gestalten
 - über alle Trainingsdatensätze der Menge iterieren
 - Größe i
5. Für genaueres Verständnis erstmal Einschub mit Gradientenverfahren

- Ziel: Gradientenvektor für bestimmten Input bestimmen:

$$\nabla J \equiv \left(\frac{\partial J}{\partial w_1}, \dots, \frac{\partial J}{\partial w_m} \right)^T.$$

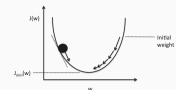


Schematic of gradient descent.

- └ Geschichtliche Entwicklung
 - └ Adeline
 - └ Gradientenverfahren

- Ziel: Gradientenvektor für bestimmten Input bestimmen:

$$\nabla J \equiv \left(\frac{\partial J}{\partial w_1}, \dots, \frac{\partial J}{\partial w_m} \right)^T.$$



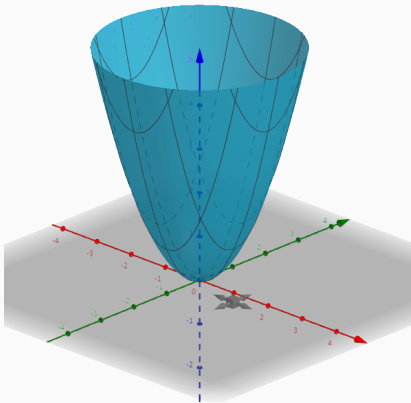
Schematic of gradient descent.

1. Wesentlicher Nachteil der Sprungfunktion: Nicht stetig & damit nicht differenzierbar
2. Adeline verwendet Identitätsfunktion
3. Abbildung erläutern, Metapher: Ball rollt Hügel herunter
 - Abbildung erstmal nur mit einem einzelnen Gewicht geplottet
 - Ableitung an einer bestimmten Stelle gleich der Steigung
 - Gradientenvektor gibt diese Richtung an
 - Mehrdimensional wenn mehreren Eingabeargumenten vorhanden
 - Steigung muss invertiert werden
4. Es folgt: Exkurs - Partielle Ableitungen

- Differenzieren von Funktionen mit mehreren Eingabewerten
- Beispiel: $z = f(x) = x^2 + y^2$

Partielle Ableitung - Notation

$$\frac{\partial \text{Abzuleitende Fkt.}}{\partial \text{Betrachtete Komponente}}$$



2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└└ Adeline

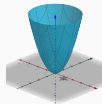
└└└ Partielle Ableitungen

Partielle Ableitungen

- Differenzieren von Funktionen mit mehreren Eingabewerten
- Beispiel: $z = f(x) = x^2 + y^2$

Partielle Ableitung - Notation

$$\frac{\partial \text{Abzuleitende Fkt.}}{\partial \text{Betrachtete Komponente}}$$

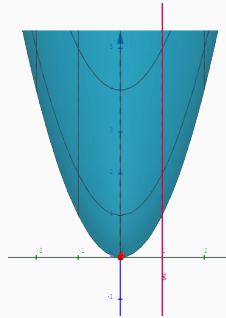
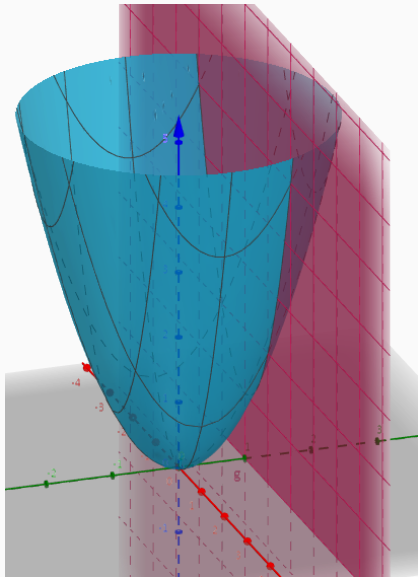


1. Notation: Bruch

- Zähler: Abzuleitende Funktion
- Nenner: Betrachtete Komponente

2. Abbildung: Fkt. geplottet mit 2 Eingabekomponenten

- Funktion: $z = f(x) = x^2 + y^2$
- Metapher: Blickwinkel erläutern
- nächste Folie miteinbeziehen



Ableitung - Beispiel

$$z = f(x, y) = x^2 + y^2$$

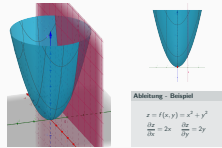
$$\frac{\partial z}{\partial x} = 2x \quad \frac{\partial z}{\partial y} = 2y$$

2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└ Adeline



Ableitung - Beispiel

$$z = f(x, y) = x^2 + y^2$$

$$\frac{\partial z}{\partial x} = 2x \quad \frac{\partial z}{\partial y} = 2y$$

1. Metapher: Blickwinkel erläutern

Gradientenverfahren

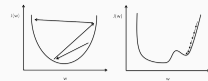
2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

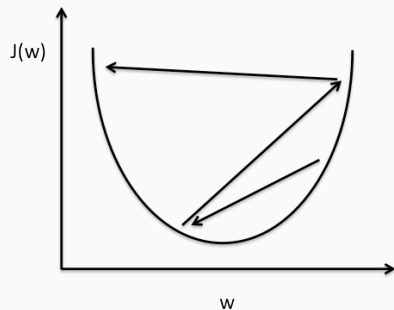
└ Adeline

└ Gradientenverfahren

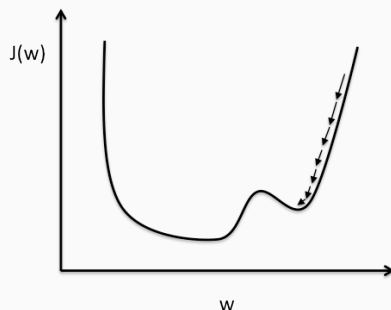


Large learning rate: Overshooting.

Small learning rate: Many iterations and convergence and trapping in local minima.

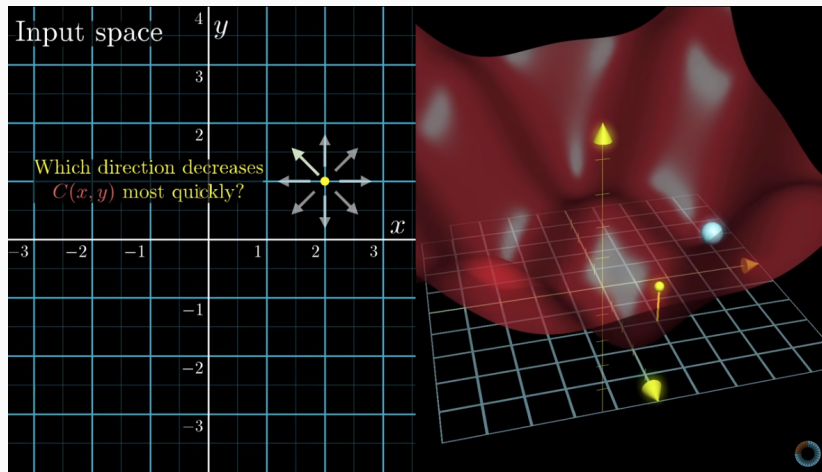


Large learning rate: Overshooting.



Small learning rate: Many iterations until convergence and trapping in local minima.

1. Lernrate kann als Schrittweite verstanden werden
2. Zwei mögliche Probleme:
 - Overshooting: Schrittweite zu groß - Minimum wird nicht erkannt
 - Lokales Minimum wird gefunden - Globales bleibt unerkannt
3. Gradientenabstieg bisher nur in 2 Dimensionen (siehe nächste Folie)



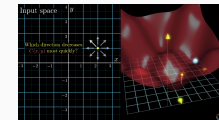
2020-04-11

Deep Learning

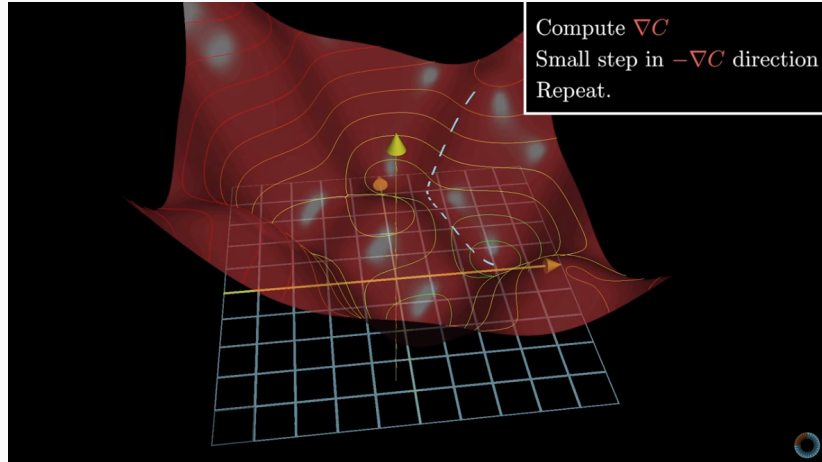
└─ Geschichtliche Entwicklung

└─ Adeline

└─ Gradientenverfahren



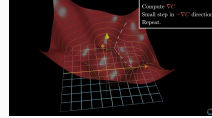
1. Abbildung: Gradientenabstieg in 3 Dimensionen geplottet
2. Hier Ball-Metapher dargestellt
3. Es folgt kompletter Durchlauf des Gradientenabsiegs



2020-04-11

Deep Learning
└─ Geschichtliche Entwicklung
 └─ Adeline
 └─ Gradientenverfahren

Gradientenverfahren



1. Abbildung: Gradientenabstieg in 3 Dimensionen geplottet
2. Hier durchgeführter Gradientenabstieg

Gradientenverfahren - Anwendung

- Gradientenvektor

$$\nabla J \equiv \left(\frac{\partial J}{\partial w_1}, \dots, \frac{\partial J}{\partial w_m} \right)^T.$$

- Allgemein: Vektorielle Darstellung

$$\Delta w = -\eta \nabla J(w)$$

- Für die jeweiligen Gewichte: Komponentenweise Darstellung

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j}$$

- Angleichung der Gewichte $w = w + \Delta w$

2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└└ Adeline

└└└ Gradientenverfahren

Gradientenverfahren

Gradientenverfahren - Anwendung

- Gradientenvektor

$$\nabla J \equiv \left(\frac{\partial J}{\partial w_1}, \dots, \frac{\partial J}{\partial w_m} \right)^T.$$

- Allgemein: Vektorielle Darstellung

$$\Delta w = -\eta \nabla J(w)$$

- Für die jeweiligen Gewichte: Komponentenweise Darstellung

$$\Delta w_j = -\eta \frac{\partial J}{\partial w_j}$$

- Angleichung der Gewichte $w = w + \Delta w$

1. Gradientenvektor: Richtung des *Abstiegs*

- mit *Nabla* dargestellt (Dreieck)
- kann auch mehrdimensional sein

2. Vektorielle Darstellung

- Eingabeparameter werden als Vektor verstanden
- mit Gradientenvektor und Negativer Lernrate verrechnet / multipliziert

3. Komponentenweise Darstellung

- negative Lernrate mit partieller Ableitung verrechnet

4. Angleichung der Gewichte:

- wie schon bei vorherigen Modellen
- Mathematische Darstellung: $w = w + \Delta w$

Kostenfunktion ableiten

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (t^{(i)} - o^{(i)})^2 \\ &= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)})^2 \\ &= \frac{1}{2} \sum_i 2(t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)}) \\ &= \sum_i (t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} \left(t^{(i)} - \sum_j w_j x_j^{(i)} \right) \\ &= \sum_i (t^{(i)} - o^{(i)}) (-x_j^{(i)})\end{aligned}$$

2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└└ Adeline

└└└ Kostenfunktion ableiten

Kostenfunktion ableiten

$$\begin{aligned}\frac{\partial J}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (t^{(i)} - o^{(i)})^2 \\ &= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)})^2 \\ &= \frac{1}{2} \sum_i 2(t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} (t^{(i)} - o^{(i)}) \\ &= \sum_i (t^{(i)} - o^{(i)}) \frac{\partial}{\partial w_j} \left(t^{(i)} - \sum_j w_j x_j^{(i)} \right) \\ &= \sum_i (t^{(i)} - o^{(i)}) (-x_j^{(i)})\end{aligned}$$

1. Ableiten der bisher vorgestellten Kostenfunktion (Least-Mean-Square)
2. Summe und Faktor vorziehen
3. Kettenregel anwenden
 - äußere Ableitung bereits bestimmt (Vorfaktor 2)
 - innere Ableitung steht noch aus
4. Faktor 2 kann vorgezogen werden, wird mit 1/2 verrechnet
5. Ursprüngliche Notation für die Ausgabe wird eingesetzt:
 - Ausgabe: $\sum_j w_j x_j^{(i)}$
6. Summe aufgelöst
 - es wird nach w_j abgeleitet
 - alle Summanden in denen dieser Faktor nicht vorkommt entfallen

2020-04-11

Deep Learning

└─ Geschichtliche Entwicklung

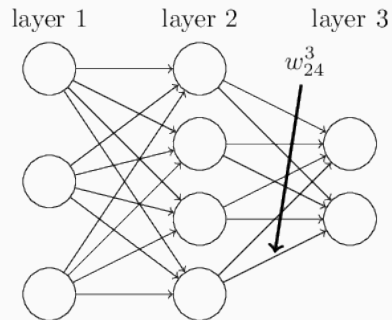
└─ Backpropagation

Geschichtliche Entwicklung

Backpropagation

Geschichtliche Entwicklung

Backpropagation



w_{jk}^l is the weight from the k^{th} neuron in the $(l-1)^{\text{th}}$ layer to the j^{th} neuron in the l^{th} layer

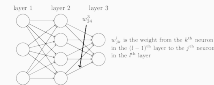
2020-04-11

Deep Learning

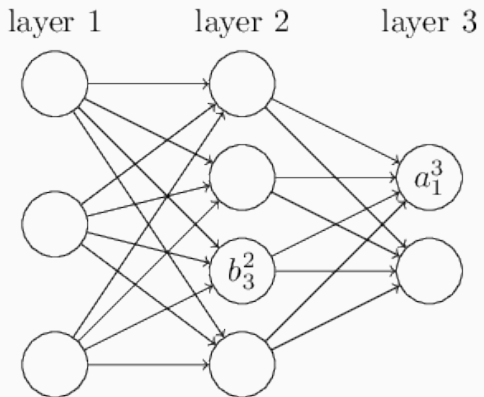
└ Geschichtliche Entwicklung

└ Backpropagation

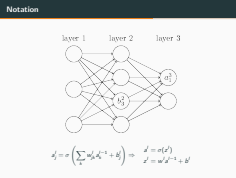
└ Notation



1. l: Exponent, steht für die Schicht
 - l - 1, weil man stets von hinten nach vorne schaut
2. Eingabe wird auch als eigene Schicht verstanden
3. j: Index Zielneuron
4. k: Index Startneuron



$$a_j^l = \sigma \left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l \right) \Rightarrow \begin{aligned} a^l &= \sigma(z^l) \\ z^l &= w^l a^{l-1} + b^l \end{aligned}$$



1. Ähnlich zu Gewichtsnotation

- l bezieht sich hierbei jedoch auf aktuelle Schicht
- j wie gehabt Index in Schicht
- Notation gilt auch für Aktivierung a

2. Wichtig: σ bezieht sich auf Vektor \Rightarrow Vektorielle Funktion

3. Jede Komponente einzeln mit σ verarbeitet

4. Abstraktion vom Ausgabewert vor der Aktivierungsfkt

- hilft später beim Ableiten

- Kostenfunktion soll minimiert werden
- Ziel: Optimale Gewichte und Schwellwerte finden
- Grobe Vorgehensweise: Iterativer Prozess
 - Fehlervektor der letzten Schicht berechnen
 - Fehler schichtweise zum Eingabelayer zurückführen
 - Parameter schichtweise nach Gradienten angleichen

2020-04-11

Deep Learning

└─ Geschichtliche Entwicklung

└─ Backpropagation

└─ Backpropagation

- Kostenfunktion soll minimiert werden
- Ziel: Optimale Gewichte und Schwellwerte finden
- Grobe Vorgehensweise: Iterativer Prozess
 - Fehlervektor der letzten Schicht berechnen
 - Fehler schichtweise zum Eingabelayer zurückführen
 - Parameter schichtweise nach Gradienten angleichen

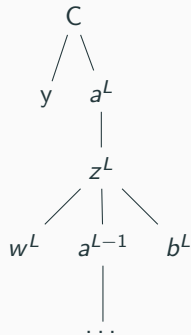
1. 1970er entwickelt, 1986 von Rumelhart, Hinton und Williams in Paper bekannt gemacht
2. Kostenfunktion wie bei Gradientenabstieg / Adeline
 - Unterschied: Hier mehrschichtiges Netz
 - Gradientenabstieg grob erläutert, ausgeblieben -
 - Anwendung im mehrschichtigen Netz und mehrdimensionale Kostenfunktion
3. Fehlervektor der letzten Schicht berechnen
 - Fehler schichtweise zum Eingabelayer zurückführen
4. Parameter schichtweise nach Gradienten angleichen

Fehler - Ausgabeschicht

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\ &= \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)\end{aligned}$$

Anmerkung: Kettenregel

$$\frac{d}{dx} [f(u)] = \frac{d}{du} [f(u)] \frac{du}{dx}$$



- **C**: Kostenfunktion
- **y**: Erwartete Ausgabe

2020-04-11

Deep Learning

- └ Geschichtliche Entwicklung
 - └ Backpropagation
 - └ Fehler - Ausgabeschicht

Fehler - Ausgabeschicht

$$\begin{aligned}\delta_j^L &= \frac{\partial C}{\partial z_j^L} \\ &= \sum_k \frac{\partial C}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} \\ &= \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)\end{aligned}$$

Anmerkung: Kettenregel

$$\frac{d}{dx} [f(u)] = \frac{d}{du} [f(u)] \frac{du}{dx}$$



- **C**: Kostenfunktion
- **y**: Erwartete Ausgabe

1. Baum nur für Netz mit einer einzigen Aktivierung
2. Zusammenhang mit Kettenregel erläutern
3. Großes L immer für Ausgabeschicht

Zusammenfassung

- Um den Fehlervektor der letzten Schicht zu bestimmen:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

- Äquivalent zu:

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

- Um die Fehler komponentenweise zu bestimmen:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$$

2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└ Backpropagation

└ Fehler - Ausgabeschicht

Fehler - Ausgabeschicht

Zusammenfassung

- Um den Fehlervektor der letzten Schicht zu bestimmen:

$$\delta^L = \nabla_a C \odot \sigma'(z^L)$$

- Äquivalent zu:

$$\delta^L = (a^L - y) \odot \sigma'(z^L)$$

- Um die Fehler komponentenweise zu bestimmen:

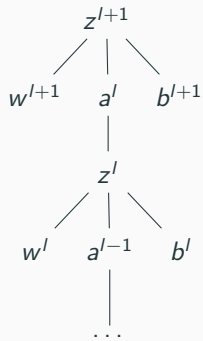
$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$$

1. $\nabla_a C$ entspricht dabei Vektor aller $\frac{\partial C}{\partial a_j^L}$ einer Schicht
2. \odot : Komponentenweise Multiplikation zweier Vektoren

Fehler - Zwischenschicht

- Zusammenhang zwischen Fehler zweier Schichten herleiten
- Es gilt: $\delta_j^l = \partial C / \partial z_j^l$ sowie $\delta_k^{l+1} = \partial C / \partial z_k^{l+1}$

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}\end{aligned}$$



2020-04-11

Deep Learning

└─ Geschichtliche Entwicklung

└─ Backpropagation

└─ Fehler - Zwischenschicht

• Zusammenhang zwischen Fehler zweier Schichten herleiten

• Es gilt: $\delta_j^l = \partial C / \partial z_j^l$ sowie $\delta_k^{l+1} = \partial C / \partial z_k^{l+1}$

$$\begin{aligned}\delta_j^l &= \frac{\partial C}{\partial z_j^l} \\ &= \sum_k \frac{\partial C}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1}\end{aligned}$$



$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

Zusammenfassung

- Komponentenweise Darstellung: $\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$
- Vektorielle Darstellung: $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$

2020-04-11

Deep Learning

└ Geschichtliche Entwicklung

└ Backpropagation

└ Fehler - Zwischenschicht

Fehler - Zwischenschicht

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1}$$
$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$$

Zusammenfassung

- Komponentenweise Darstellung: $\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l)$
- Vektorielle Darstellung: $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^l)$

Fehler - Schwellwerte & Gewichte

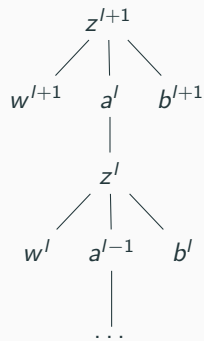
$$z_k^l = \sum_j w_{kj}^l a_j^{l-1} + b_k^l = \sum_j w_{kj}^l \sigma(z_j^{l-1}) + b_k^l$$

Schwellwerte

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l$$

Gewichte

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$



Deep Learning

Geschichtliche Entwicklung

Backpropagation

Fehler - Schwellwerte & Gewichte

2020-04-11

Fehler - Schwellwerte & Gewichte

$$z_k^l = \sum_j w_{kj}^l a_j^{l-1} + b_k^l = \sum_j w_{kj}^l \sigma(z_j^{l-1}) + b_k^l$$

Schwellwerte

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l$$

Gewichte

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$



- Menge an Trainingsdatensätzen auswählen
- Für jeden einzelnen Datensatz:
 1. **Feedforward:** Z-Wert und Aktivierung für jede Schicht $l = 2, 3, \dots, L$ berechnen.
 - Z-Wert: $z^{x,l} = w^l a^{l-1} + b^l$
 - Aktivierung $a^{x,l} = \sigma(z^l)$
 2. **Ausgabe-Fehler** $\delta^{x,L}$: Fehlervektor der Ausgangsschicht berechnen.
 - $\delta^L = \nabla_a C \odot \sigma'(z^L)$
 3. **Backpropagation-Fehler:** Rückwirkend Fehlervektor aller Schichten berechnen.
 - $\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \odot \sigma'(z^{x,l})$
- **Gradientenabstieg:** Gewichte und Schwellwerte getrennt anpassen.
 - Gewichte: $w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$
 - Schwellwerte: $b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$

2020-04-11

Deep Learning

└ Aktuelle Entwicklung

└ Convolutional Neural Network

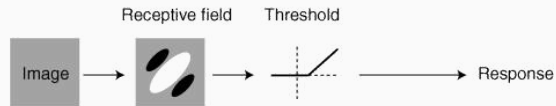
Aktuelle Entwicklung

Convolutional Neural Network

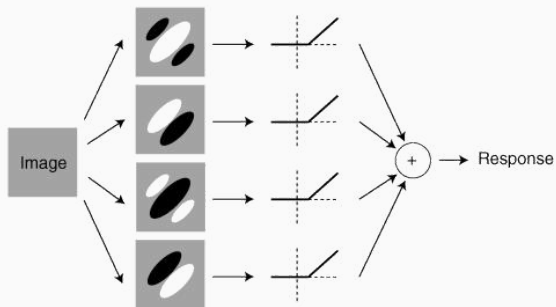
Aktuelle Entwicklung

Convolutional Neural Network

A Simple cell



B Complex cell



2020-04-11

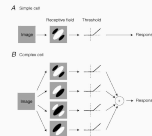
Deep Learning

└ Aktuelle Entwicklung

└ Convolutional Neural Network

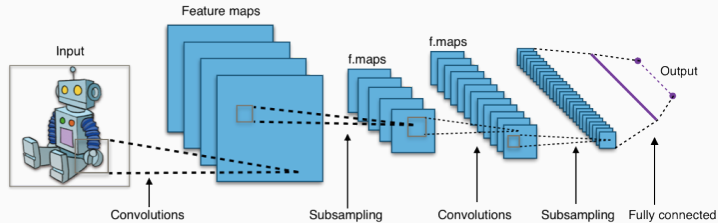
└ Biologische Zellarten

Biologische Zellarten



1. 1962: zwei Neurophysiologen Torsten Wiesel und David Hubel
2. Konzept der simple und complex cells
3. nicht positionsbunden - spatial invariance, räumliche Invarianz
4. Arten von Zellen zur Erkennung einfacher Kanten und Balken
5. *simple cells*: ist Positionsgebunden
6. *complex cells*: Muster können an beliebigen Positionen auftauchen
7. 1962: Konzept wie im Bild
8. 1980er Dr. Kunihiko Fukushima: erstes Modell nach diesem Konzept

- Yann LeCun: erstes Modell zum Erkennen von Handschrift
- *Verwendung von MNIST database of handwritten digits*
 - 60.000 Trainingsdatensätze
 - 10.000 zum Berechnen des Fehlers



2020-04-11

Deep Learning

└ Aktuelle Entwicklung

└ Convolutional Neural Network

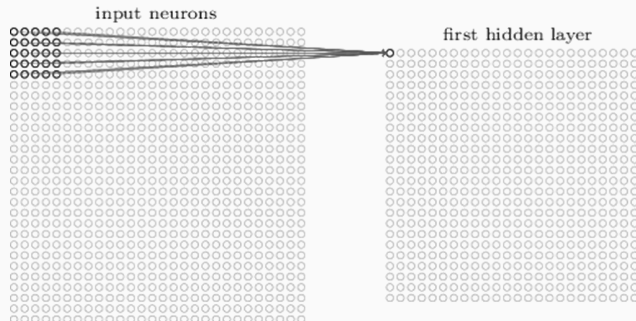
└ Anfänge



1. Pioniere, fr. Informatiker Yann LeCun
2. Bekannteste Ausarbeitung über CNN für Handschriften

Convolutinal Layer - Filter

- Mehrdimensionales Array mit Farbwerten zur Repräsentation im Rechner
- Durch Filter auf bestimmte *Low-Level* Eigenschaften schließen



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

2020-04-11

Deep Learning

└ Aktuelle Entwicklung

└ Convolutional Neural Network

└ Convolutinal Layer - Filter

Convolutinal Layer - Filter

- Mehrdimensionales Array mit Farbwerten zur Repräsentation im Rechner
- Durch Filter auf bestimmte *Low-Level* Eigenschaften schließen



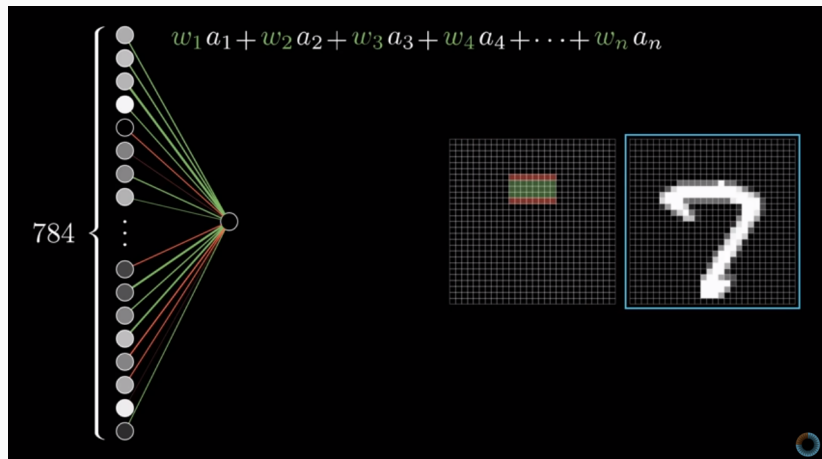
1. Farbwertearray kann pro Pixel mehrere Werte enthalten

- Generell
 - Besitzt feste Pixelgröße (*Kernelsize*) & Schrittweite
 - Scannt Bild Zeilenweise
 - *Padding* legt Verfahren für Rand des Bildes fest
 - Ausgabe wird *activation* oder *feature map* genannt
- Praxis
 - Convolutional Layer mit 32 oder 16 Bit
 - Jeder Filter generiert eigene Ausgabematrix
 - Nächster Convolutional Layer verwendet Ausgabematrizen als Input
 - Ausgabe wird in *Pooling Layer* gesteckt

- Generell
 - Besitzt feste Pixelgröße (*Kernelsize*) & Schrittweite
 - Scannt Bild Zeilenweise
 - *Padding* legt Verfahren für Rand des Bildes fest
 - Ausgabe wird *activation* oder *feature map* genannt
- Praxis
 - *Convolutional Layer* mit 32 oder 16 Bit
 - Jeder Filter generiert eigene Ausgabematrix
 - Nächster Convolutional Layer verwendet Ausgabematrizen als Input
 - Ausgabe wird in *Pooling Layer* gesteckt

1. Bsp. Filter 2×2 , Schrittweite: 2 - führt zu Halbierung der InputMatrix
2. Im Bsp. hängen immer 4 Pixel an einem Filter, die Eingabematrix wird gefaltet (convolute)

Filter - Funktionsweise



2020-04-11

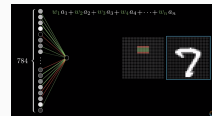
Deep Learning

└ Aktuelle Entwicklung

└ Convolutional Neural Network

└ Filter - Funktionsweise

Filter - Funktionsweise



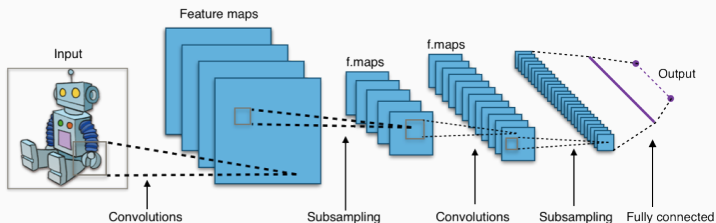
- Aggregiert die Ergebnisse von Convolutional Layern
- Ziele
 - Nur die relevantesten Signale an nächste Schicht weitergeben
 - Anzahl der Parameter im Netz reduzieren
- MaxPooling Layer am weitesten verbreitet

- Aggregiert die Ergebnisse von Convolutional Layern
- Ziele
 - Nur die relevantesten Signale an nächste Schicht weitergeben
 - Anzahl der Parameter im Netz reduzieren
- *MaxPooling Layer* am weitesten verbreitet

1. während die Größe des Inputs durch die Faltungen und das Pooling immer weiter reduziert wird, erhöht sich die Anzahl der Filter zur Erkennung von übergeordneten Signalen zunehmend

Fully Connected Layer

- Ausgangspunkt: *High-Level* Merkmale bereits durch frühere Schichten erkannt.
- Alle Neuronen der Ausgangsschicht sowie dieser Merkmale alle direkt miteinander verbunden
- Ausgabe sollte mit den richtigen Gewichten / Schwellwerten relativ eindeutige Ausgaben generieren



2020-04-11

Deep Learning

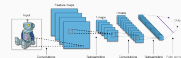
└ Aktuelle Entwicklung

└ Convolutional Neural Network

└ Fully Connected Layer

Fully Connected Layer

- Ausgangspunkt: *High-Level* Merkmale bereits durch frühere Schichten erkannt.
- Alle Neuronen der Ausgangsschicht sowie dieser Merkmale alle direkt miteinander verbunden
- Ausgabe sollte mit den richtigen Gewichten / Schwellwerten relativ eindeutige Ausgaben generieren





Deep Learning

Einführung - Thema 2

Silas Hoffmann

11. April 2020

Fachhochschule Wedel

2020-04-11

Deep Learning

└ Aktuelle Entwicklung

└ Convolutional Neural Network



Deep Learning

Einführung - Thema 2

Silas Hoffmann
11. April 2020
Fachhochschule Wedel

Sometimes, it is useful to add slides at the end of your presentation to refer to during audience questions.

The best way to do this is to include the `appendixnumberbeamer` package in your preamble and call `\appendix` before your backup slides.

metropolis will automatically turn off slide numbering and progress bars for slides in the appendix.

Backup slides

Sometimes, it is useful to add slides at the end of your presentation to refer to during audience questions.

The best way to do this is to include the `appendixnumberbeamer` package in your preamble and call `\appendix` before your backup slides.

metropolis will automatically turn off slide numbering and progress bars for slides in the appendix.

Backup Slides

Multilayer Perceptron

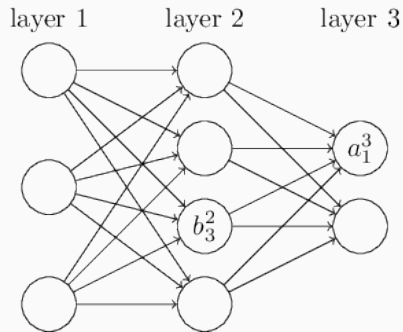
2020-04-11

Deep Learning
└ Backup Slides
 └ Multilayer Perceptron

Backup Slides
Multilayer Perceptron

Multilayer Perceptron

- In Grundzügen bereits beim Backpropagation Algorithmus erläutert



Anwendungsbereiche:

- Mustererkennung
- Funktionenapproximation
- Klassifizierung
- Prognose
- Diagnose
- Steuerung
- Optimierung

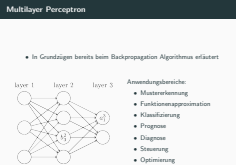
2020-04-11

Deep Learning

└ Backup Slides

└ Multilayer Perceptron

└ Multilayer Perceptron

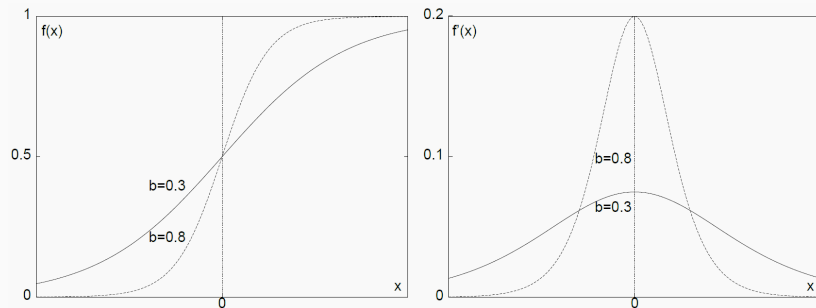


1. vielfältige Struktur
2. Mehrschichtiges Netz, bereits beim BPAlgo. verwendet
3. Zu tiefe Netz: Probleme beim Training
4. Techniken unter Deep learning zusammengefasst
5. Stuktur: mehrschichtiges vorwärtsgekoppeltes Netz
6. Feedforward: durchiterieren von Eingabewerten
- 7.
- 8.

Sigmoid Aktivierungsfunktion

- Einfach / schnell zu berechnen
- Einfach / schnell abzuleiten

$$f(x) = \frac{1}{1 + \exp(-b * x)}$$
$$f'(x) = b * f(x)(1 - f(x))$$



2020-04-11

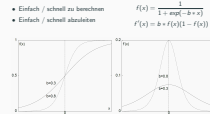
Deep Learning

└ Backup Slides

└ Multilayer Perceptron

└ Sigmoid Aktivierungsfunktion

Sigmoid Aktivierungsfunktion



1. Wertebereich: zwischen 0 und 1
2. Konstante beschreibt Steilheit der Kurve
3. Über die komplette Domäne differenzierbar

2020-04-11

Deep Learning

└ Backup Slides

└ Recurrent Neural Network

Backup Slides

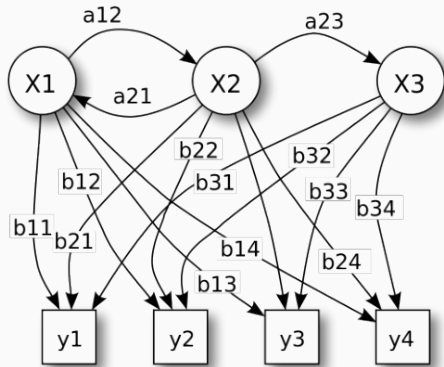
Recurrent Neural Network

Backup Slides

Recurrent Neural Network

Recurrent Neural Network

- Bekommen sowohl von Eingangssignale sowohl durch Trainingsdatensatz als auch durch Rückkopplungen.



- Mustererkennung
- Muster ergänzen
- Sprachanalyse

2020-04-11

Deep Learning

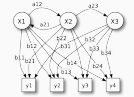
└ Backup Slides

└ Recurrent Neural Network

└ Recurrent Neural Network

Recurrent Neural Network

• Bekommen sowohl von Eingangssignale sowohl durch Trainingsdatensatz als auch durch Rückkopplungen.



- Mustererkennung
- Muster ergänzen
- Sprachanalyse

1. Bisher nur feedforward NN
2. Hierbei Reihenfolge von Bedeutung

2020-04-11

Deep Learning

└ Backup Slides

└ Recurrent Neural Network

Alle Materialien sind unter folgender URL zu finden:
https://github.com/derMacon/deeplearning_seminar

Alle Materialien sind unter folgender URL zu finden:

https://github.com/derMacon/deeplearning_seminar



3Blue1Brown - Videokurs zur Einführung in die Neuralen Netze.

https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQBObOWTQDNU6R1_67000Dx_ZCJB-3pi.

Aufgerufen am: 16-03-2020.



Übersicht - verschiedene Architekturen.

<https://www.asimovinstitute.org/neural-network-zoo/>.

Aufgerufen am: 22-03-2020.



Definition Klassifizierungsproblem.

http://ekpwww.physik.uni-karlsruhe.de/~tkuhr/HauptseminarWS1112/Keck_handout.pdf.

Aufgerufen am: 15-03-2020.

2020-04-11

Deep Learning

└ Backup Slides

└ Recurrent Neural Network

└ References

References ii



Einführung Convolutional neural network.

<https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>.

Aufgerufen am: 18-03-2020.



Öffentliche Datensätze - Übersicht.

<https://github.com/awesomedata/awesome-public-datasets>.

Aufgerufen am: 18-03-2020.



Funktionsweise - CNN.

<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1890437/>.

Aufgerufen am: 18-03-2020.



Funktionsweise - CNN.

<https://bit.ly/2QGK0Ej>.

Aufgerufen am: 18-03-2020.

2020-04-11

Deep Learning

└ Backup Slides

└ Recurrent Neural Network

└ References

References ii

Einführung Convolutional neural network.
<https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/>.

Aufgerufen am: 18-03-2020.

Öffentliche Datensätze - Übersicht.
<https://github.com/awesomedata/awesome-public-datasets>.

Aufgerufen am: 18-03-2020.

Funktionsweise - CNN.
<https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1890437/>.

Aufgerufen am: 18-03-2020.

Funktionsweise - CNN.
<https://bit.ly/2QGK0Ej>.

Aufgerufen am: 18-03-2020.



Geschichte der Convolutional neuronalen Netze.

<https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/>.

Aufgerufen am: 18-03-2020.



Khan Academy - Partielle Ableitungen (Funktion mit zwei Eingabewerten).

<https://www.youtube.com/watch?v=1CMDS4-PKKQ&t=542s>.

Aufgerufen am: 16-03-2020.



Künstliche Neuronale Netzwerke und Deep Learning - Stefan Stelle.

https://www.htwsaar.de/wiwi/fakultaet/personen/profile/selle-stefan/Selle2018e_Kuenstliche_Neuronale_Netzwerke.pdf/at_download/file.

Aufgerufen am: 24-03-2020.




2020-04-11

Deep Learning

└ Backup Slides

└ Recurrent Neural Network

└ References

-  Geschichte der Convolutional neuronalen Netze.
<https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/>.
Aufgerufen am: 18-03-2020.
-  Khan Academy - Partielle Ableitungen (Funktion mit zwei Eingabewerten).
<https://www.youtube.com/watch?v=1CMDS4-PKKQ&t=542s>.
Aufgerufen am: 16-03-2020.
-  Künstliche Neuronale Netzwerke und Deep Learning - Stefan Stelle.
https://www.htwsaar.de/wiwi/fakultaet/personen/profile/selle-stefan/Selle2018e_Kuenstliche_Neuronale_Netzwerke.pdf/at_download/file.
Aufgerufen am: 24-03-2020.



McCulloch-Pitts Neuron.

<https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1>.

Aufgerufen am: 14-03-2020.



Perceptron - Python Implementierung.

<https://github.com/rasbt/mlxtend/blob/master/mlxtend/classifier/perceptron.py>.

Aufgerufen am: 16-03-2020.



Single-Layer Neural Networks and Gradient Descent.

https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.

Aufgerufen am: 14-03-2020.



M. Nielsen.

Neural Networks and Deep Learning.

Determination Press, 2015.

2020-04-11

Deep Learning

└ Backup Slides

└ Recurrent Neural Network

└ References

- McCulloch-Pitts Neuron.
<https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1>.
Aufgerufen am: 14-03-2020.
- Perceptron - Python Implementierung.
<https://github.com/rasbt/mlxtend/blob/master/mlxtend/classifier/perceptron.py>.
Aufgerufen am: 16-03-2020.
- Single-Layer Neural Networks and Gradient Descent.
https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html.
Aufgerufen am: 14-03-2020.
- M. Nielsen.
Neural Networks and Deep Learning.
Determination Press, 2015.