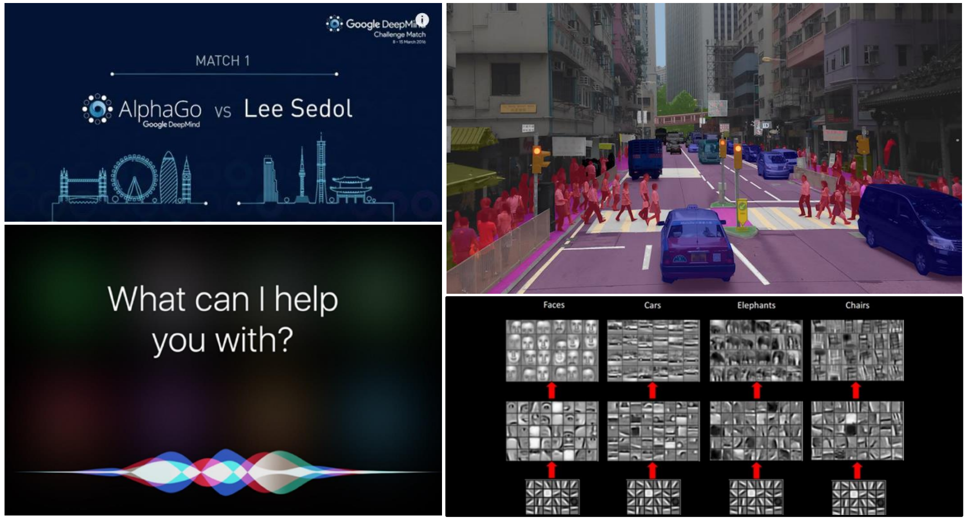
Vorlesung
Statistische Methoden der Datenanalyse
Prof. Dr. Dr. Wolfgang Rhode

Neuronale Netze



Neuronale Netze – Klassifizierung von Bildern

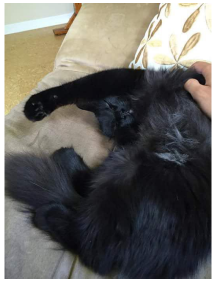
- Jedes Bild stellt für den Computer ein großes 2-dimensionales Array an Zahlen dar (3 RGB-Werte pro Pixel)
- Ziel: Reduktion des Arrays zu einem geschätzten Label aus einer festen Menge möglicher Label

Bildklassifizierung →

- 82%: Katze
- 15%: Hund
- 2%: Mütze
- 1%: Tasse

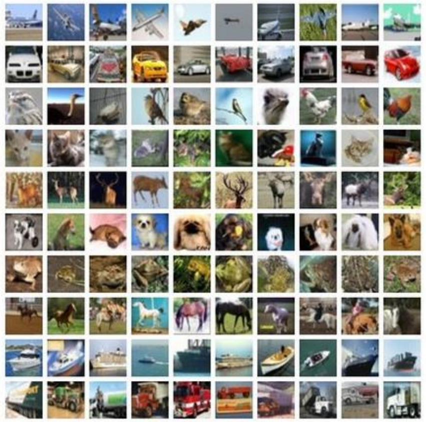
Herausforderungen bei der Klassifizierung von Bildern

- Verschiedene Ansichten
- Beleuchtungsbedingungen
- Unterschiedliche Größen
- Deformation
- Sehr ähnlicher Hintergrund
- Hohe Variation innerhalb einer Klasse



CIFAR-10 als Benchmark Datensatz

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck



Einfachster Ansatz – Nearest Neighbor Classifier

- Vergleiche Testbild mit übrigen Trainingsbildern
- Berechne die 1-Norm über alle Pixel: $d_1(I_1, I_2) = \sum_p |I_1^p - I_2^p|$
- Als Gütekriterium wird oft die Accuracy verwendet
  (Anteil richtiger Schätzungen)
- Der NN-Classifier liefert auf dem CIFAR-10 Datensatz eine Accuracy von knapp 40%
- Nachteile: Gesamter Trainingsdatensatz muss gespeichert werden,
  das Testen eines Bildes ist sehr aufwändig

Besser: Lineare Klassifikation

- Sehr einfache Auswertungs-Funktion: $f(x_i, W, b) = W x_i + b$
- x_i: Spaltenvektor, der für jeden RGB-Pixel einen Wert enthält
- $f(x_i, W, b)$ berechnet für jedes Bild x_i eine Score
  für jede Zielklasse $k = 1, \dots, K$
- *f* ist eine vektorwertige Funktion mit *K* Elementen
- Die Einträge der Matrix W werden häufig als Gewichte bezeichnet
- b heißt Bias Vektor
- W und b sind dann durch die Klassifikation zu bestimmen
- Klassifikation: $y_{\text{pred}} = \operatorname{argmax}_j (f_j)$

Lineare Klassifikation

- Jede Zeile der Matrix W ist dann ein Klassifikator für eine der Klassen
- Interpretation der linearen Klassifikation als *Template Matching*

- Jede Zeile legt ein *Prototyp* Bild der jeweiligen Klasse fest
- Der Klassifikator findet dann den am besten passenden Prototypen für das jeweilige Testbild über das Skalarprodukt

technische universität dortmund
Experimentelle Physik Vb
Astroteilchenphysik

Lineare Klassifikation – Kostenfunktion

- Das Ergebnis der linearen Klassifikation kann in die Form einer Wahrscheinlichkeitsdichte gebracht werden
- Gebräuchlichster Weg: Softmax Funktion

$$q_i(k) = \frac{\exp(f_{k,i})}{\sum_j \exp(f_{j,i})}$$

- Erzeugt für jede Klasse eine Wahrscheinlichkeit zwischen 0 und 1, sodass gilt:

$$\sum_k q_i(k) = 1$$

Prof. Dr. Dr. W. Rhode
Data-Mining – Teil 2
Statistische Methoden der Datenanalyse

technische universität dortmund
Experimentelle Physik Vb
Astroteilchenphysik

Lineare Klassifikation – Kostenfunktion

- Wie können wir die Parameter anpassen?
- Die Kostenfunktion beschreibt ein Maß dafür, wie glücklich wir mit dem Resultat der Klassifikation sind.
- Schlechte Klassifikation → Hohe Kosten
- Gute Klassifikation → Niedrige Kosten
- Gebräuchlichste Kostenfunktion zur Klassifikation: Kreuzentropie

$$H(p,q) = -\sum_k p(k) \log q(k)$$

- Liefert kleinere Werte je ähnlicher die wahre Wahrscheinlichkeitsdichte $p(x)$ zu der geschätzten Wahrscheinlichkeitsdichte $q(x)$ ist
- Problem: Wie kann diese Kostenfunktion nun optimiert werden?

Prof. Dr. Dr. W. Rhode
Data-Mining – Teil 2
Statistische Methoden der Datenanalyse

technische universität dortmund
Experimentelle Physik Vb
Astroteilchenphysik

Lineare Klassifikation – Optimierung der Kostenfunktion

- Die Kostenfunktion lässt uns die Qualität von einer speziellen Menge an Gewichten W quantifizieren
- Fragestellung: Welches W minimiert die Kostenfunktion?
- Ansatz: Dem Gradienten der Kostenfunktion folgen
- Gradient: Vektor der partiellen Ableitungen für alle Dimensionen

$$\frac{\partial f(t_i)}{\partial t_i} = \lim_{h \to 0} \frac{f(t_i + h) - f(t_i)}{h}$$

Prof. Dr. Dr. W. Rhode
Data-Mining – Teil 2
Statistische Methoden der Datenanalyse

Lineare Klassifikation – Bestimmung des Gradienten

- Analytische Berechnung:
  - Schnell
  - Exakt
- Numerische Berechnung:
  - Berechnung für jede Dimension mit einer festen Schrittweite h
  - Langsam
  - gilt nur näherungsweise
  - kann zur Überprüfung des analytischen Gradienten verwendet werden

Lineare Klassifikation – Analytische Gradientenberechnung

- Rekursive Verwendung der Kettenregel *(backpropagation)*
- Beispiel für einen Datenpunkt (x, y):
  - Netzwerk gegeben durch: $\hat{y} = f(x, a, b) = a \cdot x + b$
  - Quadratischer Abstand als Kostenfunktion: $L = (y - \hat{y})^2 = [y - f(x, a, b)]^2$

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial a} = \underbrace{2 \cdot [y - f(x, a, b)]}_{\frac{\partial L}{\partial f}} \cdot \frac{\partial f}{\partial a} = 2 \cdot [y - f(x, a, b)] \cdot x$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial b} = \overbrace{2 \cdot [y - f(x, a, b)]}^{\frac{\partial L}{\partial f}} \cdot \frac{\partial f}{\partial b} = 2 \cdot [y - f(x, a, b)]$$

Lineare Klassifikation – Analytische Gradientenberechnung

- Rekursive Verwendung der Kettenregel *(backpropagation)*
- Kostenfunktion: $H(p, q) = -\sum_k p(k) \log q(k)$
- Wahre Klassenverteilung $p(k)$: $\mathbb{1}(y = k)$ → 1 für Klasse *k*, sonst 0
- Geschätzte Klassenverteilung $q(k)$: Softmax-Funktion

$$q_i(k) = \frac{\exp(f_{k,i})}{\sum_j \exp(f_{j,i})}$$

- Mittlere Kostenfunktion für mehrere Trainings-Beispiele

$$C(f) = \frac{1}{m}\sum_{i=1}^{m}\left[-\sum_{k=1}^{K}\mathbb{1}(y_i = k)\log q_i(k)\right]$$

m : Anzahl Bilder
i : Bildnummer
K : Anzahl Klassen
k : Klassennummer

Lineare Klassifikation – Analytische Gradientenberechnung

$$C(f) = \frac{1}{m}\sum_{i=1}^{m}\hat{C}(f_i) = \frac{1}{m}\sum_{i=1}^{m}\left[-\sum_{k=1}^{K}\mathbb{1}(y_i = k)\log\frac{\exp(f_{k,i})}{\sum_j \exp(f_{j,i})}\right]$$

m : Anzahl Bilder
i : Bildnummer
K : Anzahl Klassen
k : Klassennummer

- $\hat{C}(f_i)$ zeilenweise ableiten nach $f_{a,i}$ (Score für die Klasse *a* zum Bild *i*)

$$\nabla_{f_{a,i}}\hat{C}(f_i) = -\sum_{k=1}^{K}\mathbb{1}(y_i = k)\nabla_{f_{a,i}}\log\frac{\exp(f_{k,i})}{\sum_j \exp(f_{j,i})}$$

$$\Rightarrow \nabla_{f_{a,i}}\hat{C}(f_i) = \frac{\exp(f_{a,i})}{\sum_j \exp(f_{j,i})} - \mathbb{1}(y_i = a)$$

- → Verwendung der Kettenregel um den Gradienten der anzupassenden Parameter zu bestimmen

Lineare Klassifikation – Analytische Gradientenberechnung

- Kettenregel: $\nabla_W \hat{C} = \sum_{k=1}^{K} \frac{\partial \hat{C}}{\partial f_{k,i}} \cdot \frac{\partial f_{k,i}}{\partial W}$
- Bestimmung der inneren Ableitung (nach der *a*-ten Zeile):

$$\nabla_{W_a} f_{k,i}(W,b) = \nabla_{W_a}(W_k x_i + b_k) = \delta_{ak} x_i^T$$

- Gradient für die b_k kann dann analog berechnet werden
- In der Praxis: Die Gradienten sind in ML-Bibliotheken implementiert (bspw. Tensorflow, Theano etc.)

Lineare Klassifikation – Parameteranpassung

- Anpassung der Parameter entlang der Richtung des Gradienten mit der Schrittweite h

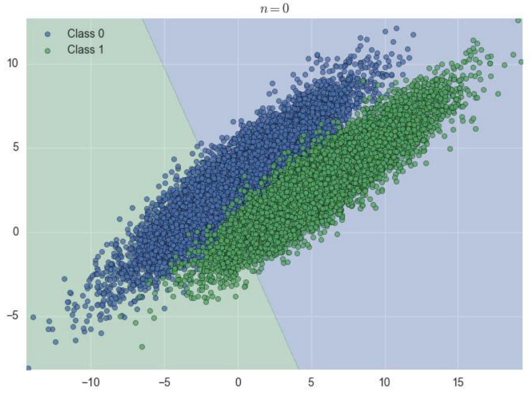
$$W^{(i+1)} = W^{(i)} - h \cdot \mathrm{grad}(W^{(i)})$$
$$b^{(i+1)} = b^{(i)} - h \cdot \mathrm{grad}(b^{(i)})$$

- Die Schrittweite wird oft als *learning rate* bezeichnet
- Dieser Prozess der wiederholten Parameteranpassung heißt *gradient descent* → iterativer Prozess

Lineare Klassifikation – *Batch Learning*

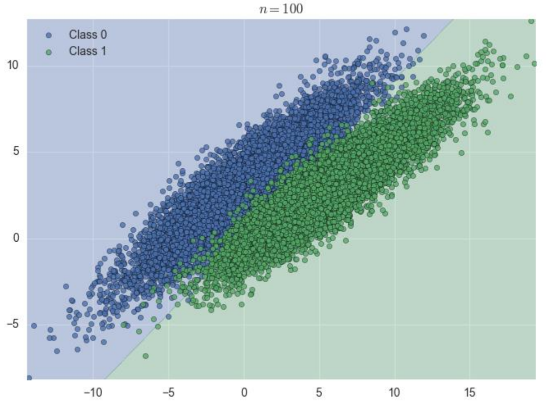
- Sind sehr viele Trainings-Daten vorhanden ist es nicht sinnvoll alle Trainings-Beispiele für jede Parameteranpassung zu verwenden
→ dauert zu lange
- Ansatz: Bestimmung des Gradienten über *batches* der Trainings-Daten
- Dann wird der Gradient nur auf einer kleinen Untermenge bestimmt, wodurch die Parameter deutlich öfter angepasst werden können

Lineare Klassifikation – Beispiel

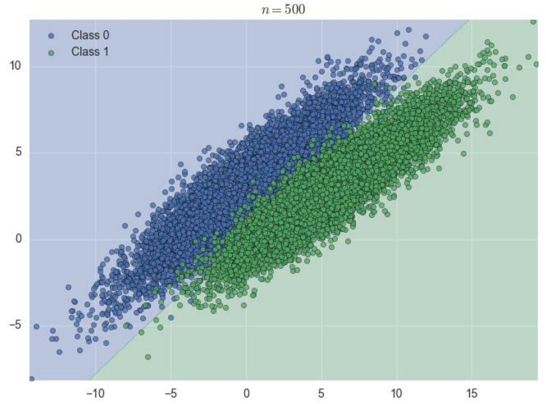


- Klassifikation nach 0 Iterationen
- Zufällige Initialisierung der Gewichte

Lineare Klassifikation – Beispiel



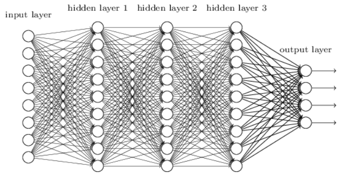
Lineare Klassifikation – Beispiel



Lineare Klassifikation – Zusammenfassung

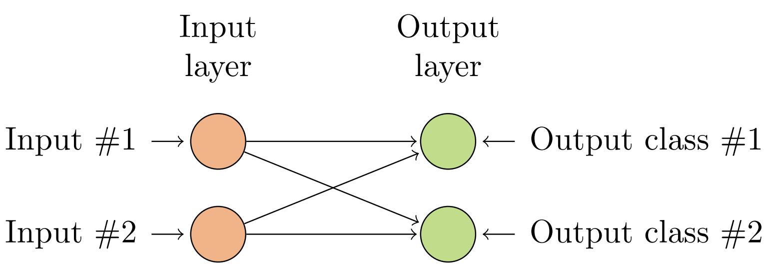
- Verwendung der Funktion $f(x_i, W, b) = W x_i + b$ um Trainings-Beispielen gegebenen Klassen zuzuordnen
- Evaluierung der Klassifikation über eine Kostenfunktion (Kreuzentropie nach der Anwendung von Softmax)
- Anpassung der Parameter W und b zur Minimierung der Kostenfunktion mittels *gradient descent*

Neuronale Netze



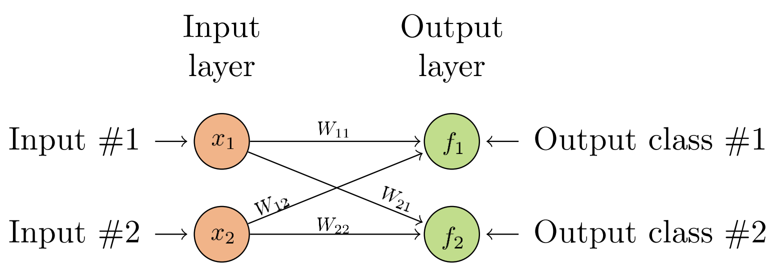
- Vorher: Lineare Klassifikation: $f = Wx$
- Jetzt: Neuronales Netz mit 2 Layern: $f = W_2\, g(W_1 x)$
- Neuronales Netz mit 3 Layern: $f = W_3\, g(W_2\, g(W_1 x))$
- Neuronale Netze setzen sich einfach aus einer beliebigen Anzahl an linearen Transformationen (Layern) mit einer nicht-linearen Aktivierungsfunktion $g(x)$ dazwischen zusammen

Neuronale Netze

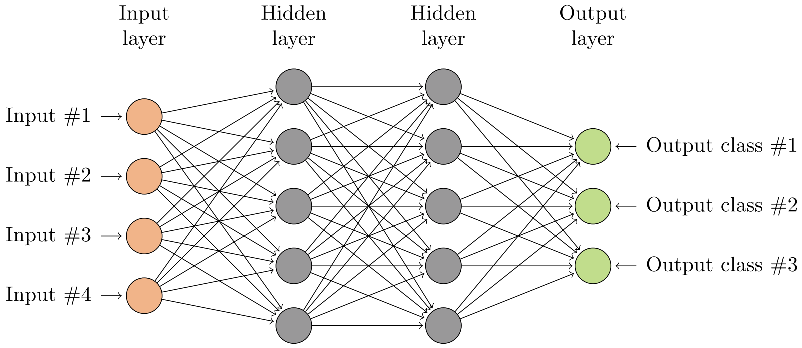


Neuronale Netze

$$f(x) = Wx = \begin{pmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \end{pmatrix}$$

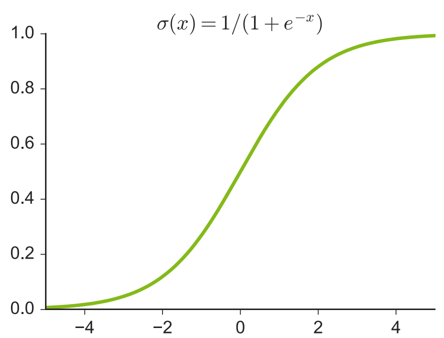


Neuronale Netze



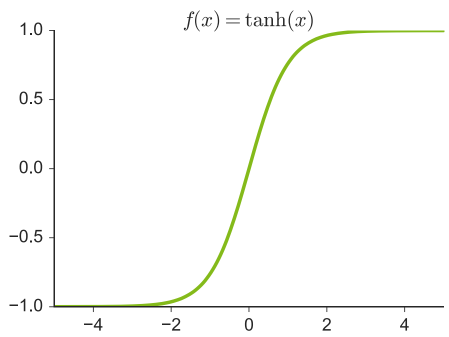
- Zahl der freien Parameter: 5 + 5 + 3 = 13 (Neuronen)
  [4 * 5] + [5 * 5] + [5 * 3] = 70 (Gewichte)
  5 + 5 + 3 = 13 (Bias Werte)
  → 83 freie Parameter

Aktivierungsfunktionen – Sigmoid



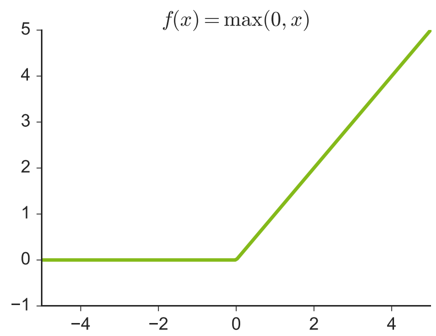
- ✓ Funktionswerte im Intervall [0, 1]
- ✓ Hat einen historischen Wert, Funktion lässt sich als saturiertes Schwellenpotential einer Nervenzelle interpretieren
- ✗ In saturierten Neuronen verschwindet der Gradient
- ✗ Werte sind nicht um Null zentriert
- ✗ Die Funktionsauswertung ist teuer

Aktivierungsfunktion – Tangens Hyperbolicus



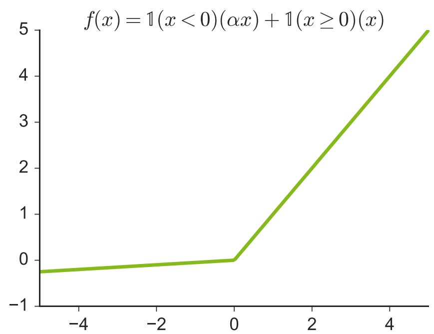
- ✓ Funktionswerte im Intervall [-1, 1]
- ✓ Lässt sich aus der Sigmoid-Funktion herleiten
- ✗ In saturierten Neuronen verschwindet der Gradient
- ✗ Die Funktionsauswertung ist teuer

Aktivierungsfunktion – Rectified Linear Unit (ReLU)



- ✓ Saturiert nicht (für positive Werte)
- ✓ Sehr effizient zu berechnen
- ✓ Neuronale Netze mit ReLU-Aktivierungsfunktionen konvergieren schneller als mit Sigmoid- oder tanh-Funktionen
- ✗ Werte nicht um Null zentriert
- ✗ Bei negativen Werten verschwindet der Gradient
  → es können keine Parameter-anpassungen mehr stattfinden

Aktivierungsfunktion – Leaky ReLU



- ✓ Saturiert nicht (für positive Werte)
- ✓ Effizient zu berechnen
- ✓ Neuronale Netze mit ReLU-Aktivierungsfunktionen konvergieren schneller als mit Sigmoid- oder tanh-Funktionen
- ✗ Werte nicht um Null zentriert

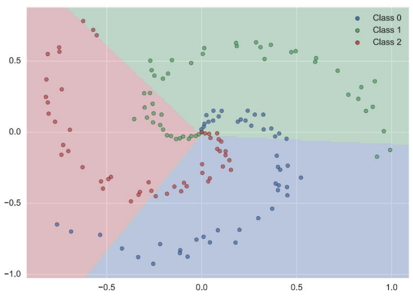
Aktivierungsfunktion – TL;DR

- In der Praxis:
  - ✓ ReLU ist fast immer eine gute Wahl
  - ✓ Leaky ReLU und tanh kann in gewissen Situationen gut funktionieren
  - ✗ Sigmoid nur in seltenen Fällen nützlich

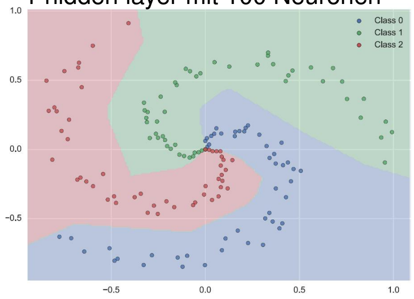
  - Wahl der Aktivierungsfunktion hängt von der Problemstellung ab
  - Aktivierungsfunktion der letzten Layer muss zum gewünschten Output passen!
- Was ist der Effekt der Nichtlinearitäten / Aktivierungsfunktionen?

Neuronales Netz – Beispiel

Lineare Klassifikation



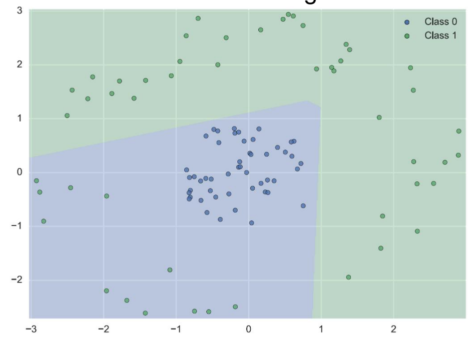
Neuronales Netz:
1 hidden layer mit 100 Neuronen



- Der Raum wird durch die Transformation und die Aktivierungsfunktion so verzerrt, dass die Spiralen durch einen gerade Schnitt im verzerrten Raum getrennt werden können

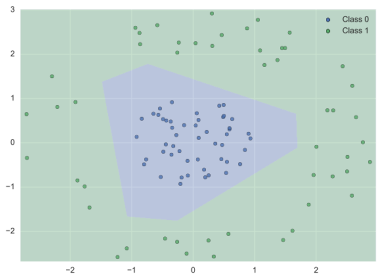
Veranschaulichung mit Kreisring-Daten

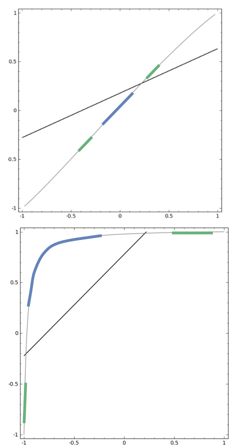
- Daten: Zwei Klassen, jeweils gleichverteilt auf einem Kreisring
- Mit zwei Neuronen im hidden layer wird eine Schneise oder ein Keil gefittet, je nach Startwerten und Verteilung der Punkte



Veranschaulichung mit Kreisring-Daten

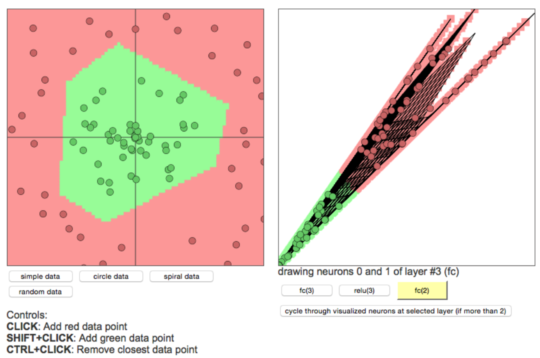
- Durch ein drittes Neuron im hidden layer wird das Problem trivial
- Durch eine Verzerrung in der dritten Dimension lassen sich die Populationen perfekt trennen





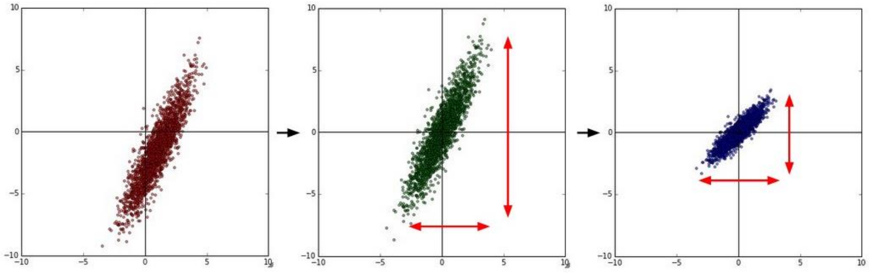
ConvnetJS demo

- http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html



Neuronale Netze – Datenvorverarbeitung bei Bildern

Ursprüngliche Daten | Ursprungszentrierung | Normalisierung



Subtraktion des Mittelwerts jedes Features

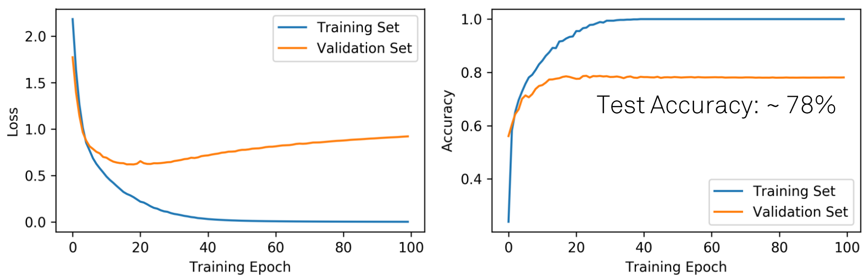
Division durch die Standardabweichung jedes Features

Neuronale Netze – Initialisierung

- Bevor der Lerner trainiert werden kann, müssen zunächst die Parameter initialisiert werden
- Initialisierung der Gewichte:
  - Erster Ansatz: Alle Gewichte mit Nullen initialisieren
    → Wenn jedes Neuron den gleichen Output berechnet, liefern alle Gradienten den gleichen Wert und der erste Schritt ist auf eine feste Richtung begrenzt
  - Besser: Initialisierung mit kleinen um Null zentrierten Zufallszahlen
    → Varianz der Gewichte hängt noch von der Anzahl der Eingangsparameter ab
- In der Praxis: Kalibration der Varianz über die Zahl der Eingangsparameter
  Für ReLU: $\mathcal{N}(0,1)\cdot\sqrt{2/n_{\text{in}}}$, sonst: $\mathcal{N}(0,1)/\sqrt{n_{\text{in}}}$
- Die Bias-Vektoren werden in der Regel mit Nullen initialisiert

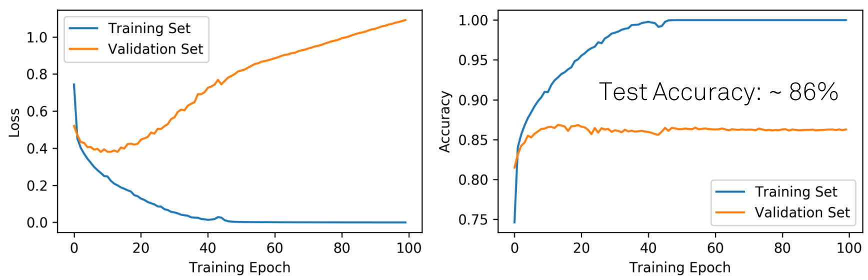
Neuronale Netze – Regularisierung

- Wie macht sich Overfitting erkennbar?
- Lösungsansätze:
  Trainingsdatensatz vergrößern, Model verkleinern, Regularisierung



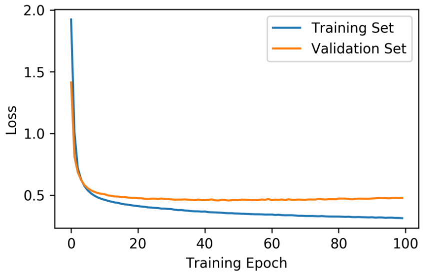
Neuronale Netze – Regularisierung

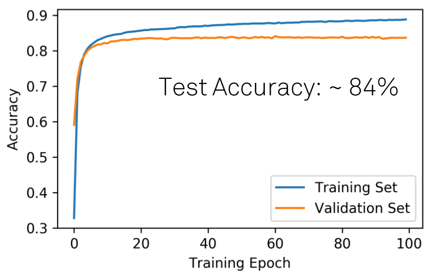
- Vergrößere Trainingsdatensatz:



Neuronale Netze – Regularisierung

- Verringere Modelgröße:





Neuronale Netze – Regularisierung

- Regularisierung kann verwendet werden um Overfitting zu verhindern

- L2 Regularisierung:
  - Meist genutzte Form der Regularisierung
  - Einführung des Strafterms $\frac{1}{2}\lambda w^2$ in die Kostenfunktion für jedes Gewicht w → Modifikation des Gradienten mit dem Term λw
  - Interpretation: Die Kostenfunktion bevorzugt diffuse Gewichtsmatrizen

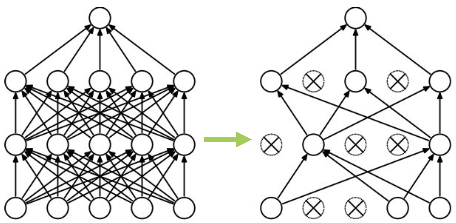
- L1 Regularisierung:
  - Der Strafterm nimmt hier die Form $\lambda\,|w|$ an
  - Sorgt ebenso im Allgemeinen für kleinere Gewichte
  - Bevorzugt große Gewichte an wichtigen Verbindungen im Netz und lässt die restlichen Gewichte gegen Null gehen → Ignoriert Rauschen in den Daten

Neuronale Netze – Regularisierung

- Dropout Regularisierung

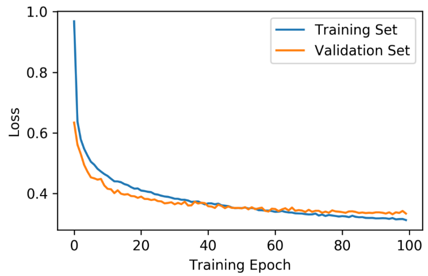
- Neuronen werden nur mit einer Wahrscheinlichkeit p während des Trainings aktiviert

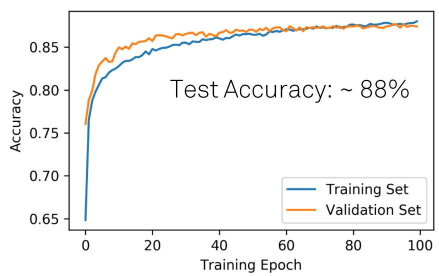
- Die Wahrscheinlichkeit p ist dabei ähnlich zur Regularisierungsstärke λ



Neuronale Netze – Regularisierung

- Dropout Regularisierung:





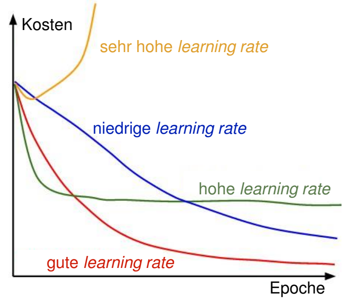
Neuronale Netze – Training

Trainings-Checkliste:

- Überprüfung des Gradienten mit dem numerischen Gradienten
- Liefert die Kostenfunktion nach der Initialisierung den Wert, der erwartet wird, wenn das Netz rät
- Erhöhung der Regularisierungsstärke λ sollte die anfänglichen Kosten erhöhen
- Ein kleines Subset der Daten sollte perfekt klassifizierbar sein

Neuronale Netze – Training

- Wichtig: Kontrolle der Kostenfunktion während des Trainings



- Niedrige *learning rate:* langsamer, aber stetiger Fortschritt
- Hohe *learning rate:* Kosten nehmen sehr schnell ab, können um das wahre Minimum oszillieren

→ Lösung: Variation der *learning rate* mit der Zeit *(learning rate decay)*

- Kosten verändern sich gar nicht → *learning rate* ist zu niedrig
- Kosten werden *NaN* → *learning rate* ist oft zu hoch

Neuronale Netze – Parameteranpassung

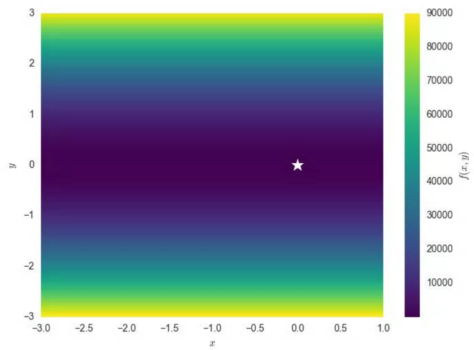
- Anpassung der Parameter entlang der Richtung des Gradienten mit der *learning rate h*

$$W^{(i+1)} = W^{(i)} - h \cdot \mathrm{grad}(W^{(i)})$$
$$b^{(i+1)} = b^{(i)} - h \cdot \mathrm{grad}(b^{(i)})$$

- Gleiche Parameteranpassung wie bei der linearen Klassifikation

Neuronale Netze – Parameteranpassung

- Gradientenabsteig in $f(x,y) = a_1 x^2 + a_2 y^2 \quad \text{mit } a_1 = 0.5, a_2 = 100$



→ Das Minimum zu finden dauert sehr lange

Neuronale Netze – Parameteranpassung

Momentum Update:

- Physikalische Herangehensweise an das Optimierungsproblem
- Interpretation der Kosten als Höhe in einem Terrain
- Das Optimierungsproblem wird dann äquivalent zur Simulation eines Teilchens, das sich durch die Kosten-Landschaft bewegt

$$E_{\text{pot}} = mgh$$
$$F = -\nabla E_{\text{pot}}$$

- Der Gradient ist proportional zur Beschleunigung des Teilchens

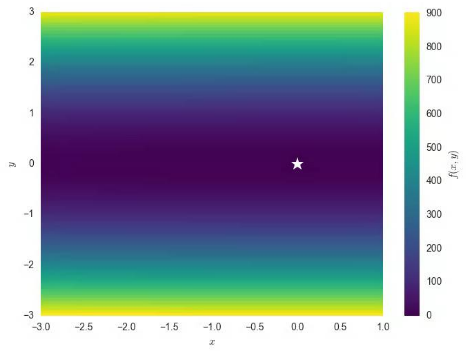
$$v^{(i+1)} = v^{(i)} \cdot \mu - h \cdot \text{grad}(W^{(i)})$$

- Dabei ist µ der Reibungskoeffizient (muss beim Training optimiert werden)

$$W^{(i+1)} = W^{(i)} + v^{(i+1)}$$

Neuronale Netze – Parameteranpassung

- Momentum Update mit µ = 0,8



In Richtungen mit einem über einige Schritte konsistenten Gradienten baut sich eine Geschwindigkeit auf.

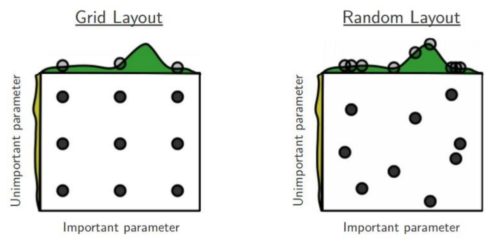
Neuronale Netze – Optimierung von Hyperparametern

- Hyperparameter sind alle freien Parameter, die nicht vom Modell angepasst werden
  - Anfängliche *learning rate* h
  - Änderung der *learning rate* mit der Zeit
  - Regularisierungsstärke λ bzw. die Dropout-Wahrscheinlichkeit *p*
  - Reibungskoeffizient μ bei der Parameteranpassung

- Wichtig: Validierung
  - Lange Trainingslaufzeit bei großen neuronalen Netzen
    → Kreuzvalidierung oft nicht praktikabel
  - Aufteilen der Daten in Trainingsset, Testset und Validierungsset
    → Optimierung der Hyperparameter auf dem Validierungsset
    → Final: Bestimmung der Accuracy auf dem Testset

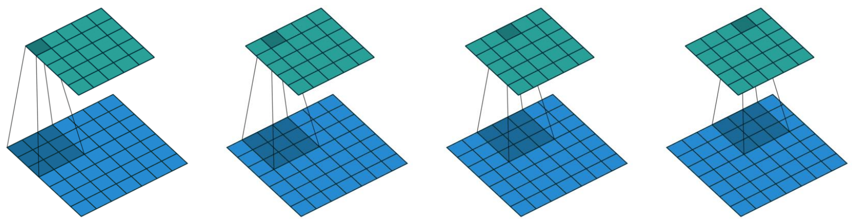
Neuronale Netze – Optimierung von Hyperparametern

- Hyperparameter Scans sollten zunächst grob auf logarithmischen Skalen durchgeführt werden
- Die besten Hyperparameter-Werte liegen am Rand → Bereich erweitern
- In der richtigen Größenordnung kann dann noch eine feinere Suche stattfinden
- Random Search sollte einer Grid Search vorgezogen werden



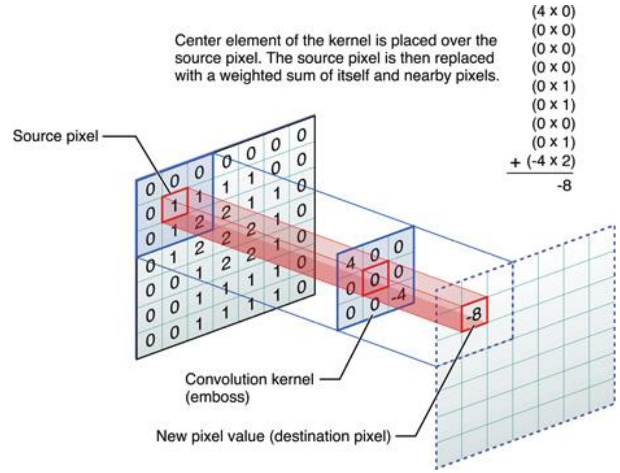
Neuronale Netze – Convolutional Neural Networks (CNNs)

- Fully-connected / dense layer in der Bilderkennung: jeder Pixel ist mit jedem Pixel aus der vorhergehenden Ebene verbunden
  → Führt zur Explosion der Anzahl freier Parameter!
- Lösung: Nutze Translationsinvarianz (es ist egal, wo die Katze im Bild ist) und verbinde Pixel nur mit Pixel in der lokalen Umgebung



technische universität dortmund
Experimentelle Physik Vb
Astroteilchenphysik

Neuronale Netze – Convolutional Neural Networks (CNNs)



Prof. Dr. Dr. W. Rhode
Data-Mining – Teil 2
Statistische Methoden der Datenanalyse

technische universität dortmund
Experimentelle Physik Vb
Astroteilchenphysik

Neuronale Netze – Convolutional Neural Networks (CNNs)

0	0	0
0	1	0
0	0	0

-1	-1	-1
-1	8	-1
-1	-1	-1

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

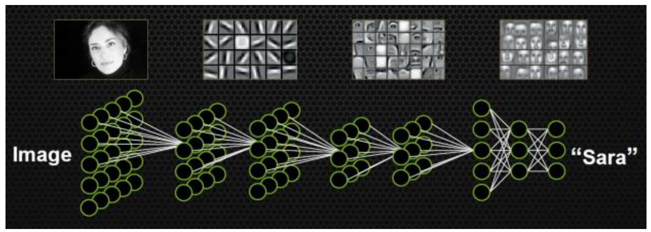
0	-1	0
-1	4	-1
0	-1	0

Prof. Dr. Dr. W. Rhode
Data-Mining – Teil 2
Statistische Methoden der Datenanalyse

technische universität dortmund
Experimentelle Physik Vb
Astroteilchenphysik

Neuronale Netze – State of the Art

- Convolutional Neural Nets zur Bildklassifizierung



- ~97% Accuracy auf dem CIFAR-10 Datensatz
- ~75% Accuracy auf dem CIFAR-100 Datensatz
- $\mathcal{O}(10^8)$ freie Parameter
- Werden auch viel im Bereich Reinforcement Learning genutzt

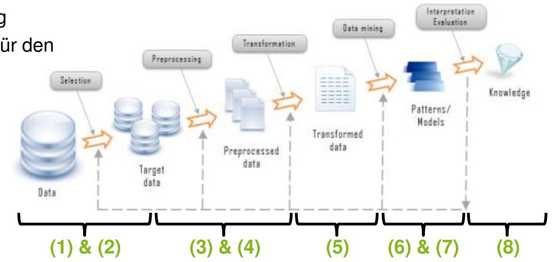
Prof. Dr. Dr. W. Rhode
Data-Mining – Teil 2
Statistische Methoden der Datenanalyse

Neuronale Netze – Zusammenfassung

- Verkettung von linearen Transformationen mit Nichtlinearitäten
  → Lineare Klassifikation der Repräsentation nach letztem Hidden Layer
- Minimierung einer Kostenfunktion mittels Gradientenabstieg
  - Klassifikation: Kreuzentropie
  - Regression: χ^2
- Initialisierung und geeignete Wahl der Architektur/Hyperparameter wichtig
- Kann auf sehr rohe Daten angewandt werden
  (Repräsentation durch trennstarke Attribute nicht zwingend notwendig)
- Hohe Dimensionalität → schwer zu interpretieren

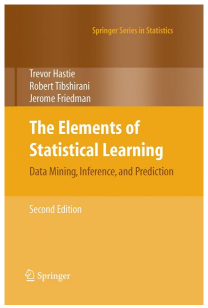
Data Mining Prozesse

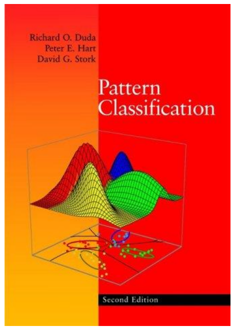
(1) Definition der Ziele der Wissensfindung
(2) Bereitstellung von Hintergrundwissen für den jeweiligen Fachbereich
(3) Datenauswahl
(4) Datenbereinigung
(5) Datenreduktion und -transformation
(6) Auswahl eines Modells
(7) Data-Mining
(8) Interpretation

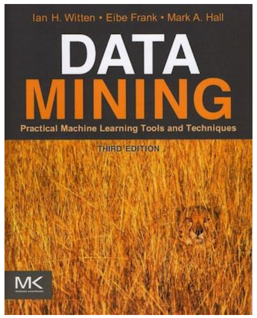


- Bei der Auswahl eines Modells und dem anschließenden Schritt des „Data-Minings“ werden eine Vielzahl von Algorithmen angewendet und optimiert, um den stärksten Algorithmus für das gestellte Problem zu finden
- Unabdingbar ist alle Algorithmen auf ihre Stabilität zu untersuchen und die gewonnenen Qualitätsparameter und Ergebnisse zu validieren

Literatur







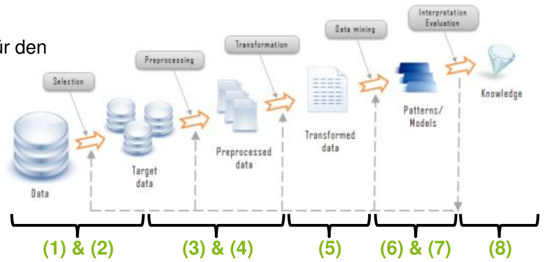
Online unter:
http://statweb.stanford.edu/~tibs/ElemStatLearn/

Werkzeugkästen

scikit learn machine learning in Python
rapidminer
TMVA
WEKA The University of Waikato
TensorFlow™

Interpretation der Ergebnisse des KDD/Data-Mining-Prozesses

- (1) Definition der Ziele der Wissensfindung
- (2) Bereitstellung von Hintergrundwissen für den jeweiligen Fachbereich
- (3) Datenauswahl
- (4) Datenbereinigung
- (5) Datenreduktion und -transformation
- (6) Auswahl eines Modells
- (7) Data-Mining
- (8) Interpretation



- Typische Techniken für die Interpretation der gewonnen Ergebnisse sind:
  - Schätzen von Größen und ihrer Konfidenzbereiche
  - Fitten von Funktionen
  - Testen von Hypothesen
  - Entfalten der Daten