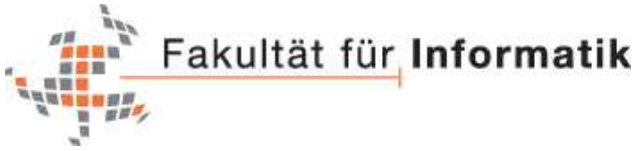
Info IV Tutorium

Sondertutorium

Benjamin Vogel & Thomas Pajor



IBDS Prautzsch

27. Juli 2007

Erbauliches zum Jahr der Geisteswissenschaften

Es reicht nicht aus, einen gut funktionierenden Verstand zu haben; das Wichtigste ist, ihn auch gut zu nutzen.

René Descartes (1596-1650)

Frage

Könntet ihr nochmal was zum
Boyer-Moore-Algorithmus machen?

Boyer-Moore

- Muster um möglichst viele Stellen weiterschieben
- Nutzt Struktur des Musters in Pre-Processing-Phase $\rightarrow$ Sprungtabellen
- Vergleich von rechts nach links
- 2 Strategien: “Schlechtes-Zeichen” und “Gutes Ende”
- Auswahl der Strategie mit der maximalsten Verschiebedistanz
- $\mathcal{O}(n/m)$

Boyer & Moore Algorithmus

Algorithmus 1 : BoyerMoore

Eingabe : Text α und Muster β

Ausgabe : Erstes Vorkommen von β in α

```
1 ρ ← BadCharacter(β)
2 σ ← StrongSuffix(β)
3 für i ← 1 ... n − m tue
4     für j ← m ... 1 tue
5         wenn β_j ≠ α_{i+j−1} dann
6             δ ← max{j − ρ(α_i), j − σ(j)}
7             i ← i + δ
8             break
9         Muster gefunden an Stelle i
```

Schlechtes Zeichen

- Abbildung: $\rho : \Sigma \rightarrow \mathbb{Z}$
- Mismatch an Stelle j an Zeichen $c \notin \beta \Rightarrow \rho(c) = 0$
- Mismatch an Stelle j an Zeichen $c \in \beta \Rightarrow \rho(c)$ rechtestes Vorkommen von $c \in \beta$.
- Neue Vergleichsposition: $i = i + (m - j) + (j - \rho(\alpha_i))$
- Einfacher BM-Algorithmus (Manuskript - Seiten 1 und 2) verwendet nur die "Schlechtes Zeichen"-Strategie

Bad-Character-Rule

Algorithmus 2 : BadCharacter

Eingabe : Muster β

Ausgabe : Sprungfunktion $\rho : \Sigma \to \mathbb{Z}$

```
1 für alle c ∈ Σ tue                    /* Alle Werte mit 0 initialisieren */
2   ρ(c) ← 0

3 für j = m...1 tue                     /* Rechteste Vorkommen in β ermitteln */
4   wenn ρ(β_j) = 0 dann
5     ρ(β_j) ← j
```

Gutes Ende

- Abbildung: $\sigma : \{1, \ldots, m\} \to \mathbb{Z}$
- 1. Fall: Mismatch an Position j, aber Suffix $\beta_{j+1,\ldots,m}$ nochmal in β.
  Gesucht: rechtestes Vorkommen des Suffix $\beta_{j+1,\ldots,m}$ in β, so dass "das Zeichen davor" nicht mit β_j übereinstimmt
- Formal: größtmögliches k, so dass $\beta_{k+1,\ldots,k+m-j-1} = \beta_{j+1,\ldots,m}$ mit $\beta_k \neq \beta_j$.
  Verschiebe um $j - k \rightsquigarrow \sigma(j) = k$

Gutes Ende

- Abbildung: $\sigma : \{1, \ldots, m\} \to \mathbb{Z}$
- 1. Fall: Mismatch an Position j, aber Suffix $\beta_{j+1,\ldots,m}$ nochmal in β.
  Gesucht: rechtestes Vorkommen des Suffix $\beta_{j+1,\ldots,m}$ in β, so dass "das Zeichen davor" nicht mit β_j übereinstimmt
- Formal: größtmögliches k, so dass $\beta_{k+1,\ldots,k+m-j-1} = \beta_{j+1,\ldots,m}$ mit $\beta_k \neq \beta_j$.
  Verschiebe um $j - k \rightsquigarrow \sigma(j) = k$
- 2. Fall: Mismatch an Position j, aber Suffix $\beta_{j+1,\ldots,m}$ nicht nochmal in β bzw. zwar in β, aber "das Zeichen davor" stimmt mit β_j überein.
  Aber: Ende vom Suffix ist zugleich Präfix vom Muster
- Formal: großes k derart, dass $\beta_{m-k+1,\ldots,m} = \beta_{1,\ldots,k}$
  Verschiebe um $m - k \rightsquigarrow \sigma(j) = k + j - m$

Gutes Ende

- Abbildung: $\sigma : \{1, \ldots, m\} \to \mathbb{Z}$
- 1. Fall: Mismatch an Position j, aber Suffix $\beta_{j+1,\ldots,m}$ nochmal in β.
  Gesucht: rechtestes Vorkommen des Suffix $\beta_{j+1,\ldots,m}$ in β, so dass "das Zeichen davor" nicht mit β_j übereinstimmt
- Formal: größtmögliches k, so dass $\beta_{k+1,\ldots,k+m-j-1} = \beta_{j+1,\ldots,m}$ mit $\beta_k \neq \beta_j$.
  Verschiebe um $j - k \rightsquigarrow \sigma(j) = k$
- 2. Fall: Mismatch an Position j, aber Suffix $\beta_{j+1,\ldots,m}$ nicht nochmal in β bzw. zwar in β, aber "das Zeichen davor" stimmt mit β_j überein.
  Aber: Ende vom Suffix ist zugleich Präfix vom Muster
- Formal: großes k derart, dass $\beta_{m-k+1,\ldots,m} = \beta_{1,\ldots,k}$
  Verschiebe um $m - k \rightsquigarrow \sigma(j) = k + j - m$
- Sonderfall: $j = m \rightsquigarrow \sigma(j) = m - 1$

Strong-Suffix-Rule

Algorithmus 3 : StrongSuffix

Eingabe : Muster β, $|\beta| = m$

Ausgabe : Sprungfunktion $\sigma : \{1, \ldots, m\} \to \mathbb{Z}$

```
// Sonderfall
```

1 $\sigma(m) \leftarrow m - 1$

2 **für** $j \leftarrow (m-1) \ldots 1$ **tue**

3 $\quad S \leftarrow \beta_{j+1} \ldots \beta_m$

$\quad$ `// Fall I`

4 $\quad$ **wenn** $k \leftarrow \max\{\kappa < j \mid \beta_{\kappa+1} \ldots \beta_{\kappa+m-j-1} = S \wedge \beta_\kappa \neq \beta_j\} \neq \perp$ **dann**

5 $\quad\quad \sigma(j) \leftarrow k$

$\quad$ `// Fall II`

6 $\quad$ **sonst**

7 $\quad\quad k \leftarrow \max\left(\{\kappa \mid \beta_1 \ldots \beta_\kappa = \beta_{m-\kappa+1} \ldots \beta_m\} \cup \{0\}\right)$

8 $\quad\quad \sigma(j) \leftarrow k + j - m$

Die Tabellen (I)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Bad Character (rechtestes Vorkommen von c in β)

c	A	B	N	sonstige

Die Tabellen (I)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Bad Character (rechtestes Vorkommen von c in β)

c	A	B	N	sonstige
$\rho(c)$	7	4	6	0

Die Tabellen (II)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Strong Suffix

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
k							-
Fall							3
$\sigma(j)$							6

Die Tabellen (II)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Strong Suffix

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
k						4	-
Fall						1	3
$\sigma(j)$						4	6

Die Tabellen (II)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Strong Suffix

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
k					1	4	-
Fall					2	1	3
$\sigma(j)$					-1	4	6

Die Tabellen (II)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Strong Suffix

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
k				1	1	4	-
Fall				2	2	1	3
$\sigma(j)$				-2	-1	4	6

Die Tabellen (II)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Strong Suffix

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
k			1	1	1	4	-
Fall			2	2	2	1	3
$\sigma(j)$			-3	-2	-1	4	6

Die Tabellen (II)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Strong Suffix

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
k		1	1	1	1	4	-
Fall		2	2	2	2	1	3
$\sigma(j)$		-4	-3	-2	-1	4	6

Die Tabellen (II)

Muster

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A

$\rightsquigarrow m = 7$

Strong Suffix

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
k	1	1	1	1	1	4	-
Fall	2	2	2	2	2	1	3
$\sigma(j)$	-5	-4	-3	-2	-1	4	6

Beispiel

c	A	B	N	sonstige
$\rho(c)$	7	4	6	0

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
$\sigma(j)$	-5	-4	-3	-2	-1	4	6

A N A N A S B A N A N A B A N A N E

A N A B A N A

Beispiel

c	A	B	N	sonstige
$\rho(c)$	7	4	6	0

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
$\sigma(j)$	-5	-4	-3	-2	-1	4	6

A N A N A S B A N A N A B A N A N E

A N A B A N A

j-BC(B) = 7-4 = 3; j-SS(7) = 7-6 = 1

Beispiel

c	A	B	N	sonstige
$\rho(c)$	7	4	6	0

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
$\sigma(j)$	-5	-4	-3	-2	-1	4	6

A N A N A S B A N A N A B A N A N E

A N A B A N A

A N A B A N A

j-BC(B) = 7-4 = 3; j-SS(7) = 7-6 = 1

Beispiel

c	A	B	N	sonstige
$\rho(c)$	7	4	6	0

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
$\sigma(j)$	-5	-4	-3	-2	-1	4	6

A N A N A S B A N A N A B A N A N E

A N A B A N A

A N A B A N A

j-BC(B) = 7-4 = 3; j-SS(7) = 7-6 = 1

j-BC(S) = 3-0 = 3; j-SS(3) = 3-(-3) = 6

Beispiel

c	A	B	N	sonstige
$\rho(c)$	7	4	6	0

j	1	2	3	4	5	6	7
β	A	N	A	B	A	N	A
$\sigma(j)$	-5	-4	-3	-2	-1	4	6

A N A N A S B A N A N A B A N A N E

A N A B A N A

A N A B A N A

A N A B A N A

j-BC(B) = 7-4 = 3; j-SS(7) = 7-6 = 1

j-BC(S) = 3-0 = 3; j-SS(3) = 3-(-3) = 6

Frage

Könntet ihr noch was zum Vergleich des algebraischen und geometrischen Simplexverfahrens machen?

Simplex-Algorithmus

Das Simplex-Verfahren ist ein Algorithmus zur Lösung linearer Programme.

Simplex-Algorithmus

Das Simplex-Verfahren ist ein Algorithmus zur Lösung linearer Programme.

Definition (Lineares Programm)

Ein *Lineares Programm* besteht aus einer linearen Zielfunktion

$$c_1 x_1 + c_2 x_2 + \ldots + c_n x_n$$

die maximiert/minimiert werden soll, sowie m Ungleichungen der Form

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & \begin{matrix}\leq\\=\\\geq\end{matrix} & b_1 \\ \vdots & & & & \ddots & + & & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & \begin{matrix}\leq\\=\\\geq\end{matrix} & b_m \end{array}$$

Simplex-Algorithmus

Das Simplex-Verfahren ist ein Algorithmus zur Lösung linearer Programme.

Definition (Lineares Programm)

Ein *Lineares Programm* besteht aus einer linearen Zielfunktion

$$c^\top x$$

die maximiert/minimiert werden soll, sowie m Ungleichungen der Form

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & \begin{matrix}\leq\\=\\\geq\end{matrix} & b_1 \\ \vdots & & & & \ddots & + & & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & \begin{matrix}\leq\\=\\\geq\end{matrix} & b_m \end{array}$$

Simplex-Algorithmus

Das Simplex-Verfahren ist ein Algorithmus zur Lösung linearer Programme.

Definition (Lineares Programm)

Ein *Lineares Programm* besteht aus einer linearen Zielfunktion

$$c^\top x$$

die maximiert/minimiert werden soll, sowie m Ungleichungen der Form

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & \leq & b_1 \\ \vdots & & & & \ddots & + & & & \vdots \\ a_{m1}x_1 & + & a_{m2}x_2 & + & \cdots & + & a_{mn}x_n & \leq & b_m \end{array}$$

Simplex-Algorithmus

Das Simplex-Verfahren ist ein Algorithmus zur Lösung linearer Programme.

Definition (Lineares Programm)

Ein *Lineares Programm* besteht aus einer linearen Zielfunktion

$$c^\top x$$

die maximiert/minimiert werden soll, sowie m Ungleichungen der Form

$$Ax \leq b$$

Simplex-Algorithmus

Das Simplex-Verfahren ist ein Algorithmus zur Lösung linearer Programme.

Definition (Lineares Programm)

Ein *Lineares Programm* besteht aus einer linearen Zielfunktion

$$c^\top x$$

die maximiert/minimiert werden soll, sowie m Ungleichungen der Form

$$Ax \leq b$$

Dabei sind $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$ und $b \in \mathbb{R}^m$.

Simplex-Algorithmus

Je nach Verfahren (geometrisch oder algebraisch) ist eine andere *Ausgangsform* des LP notwendig.

Simplex-Algorithmus

Je nach Verfahren (geometrisch oder algebraisch) ist eine andere *Ausgangsform* des LP notwendig.

Geometrisch

Definition (Standardform)

Gegeben: $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ und $c \in \mathbb{R}^n$.

Gesucht: Lösung $x \in \mathbb{R}^n$, so dass

$$c^\top x$$

maximiert unter

$$Ax \leq b \quad \text{und} \quad x \geq 0$$

Algebraisch

Definition (Normalform)

Gegeben: $v \in \mathbb{R}^n$, $c \in \mathbb{R}^m$, $B \in \mathbb{R}^{m \times n}$.

Gesucht: Lösung $x \in \mathbb{R}^n$, so dass

$$z = c^\top x + v$$

maximiert unter

$$y = Bx + b \quad \geq 0$$

Simplex-Algorithmus (Geometrisch)

Das Vorgehen anschaulich:

Simplex-Algorithmus (Geometrisch)

Das Vorgehen anschaulich:

- Jede Nebenbedingung der Form

$$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i$$

beschreibt einen *Halbraum* im $\mathbb{R}^n$.

Simplex-Algorithmus (Geometrisch)

Das Vorgehen anschaulich:

- Jede Nebenbedingung der Form

$$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i$$

  beschreibt einen *Halbraum* im $\mathbb{R}^n$.
- Da *alle* Nebenbedingung erfüllt sein müssen definiert der *Schnitt* der Nebenbedingungen die Menge aller Lösungen – ein *konvexer Polyeder*.

Simplex-Algorithmus (Geometrisch)

Das Vorgehen anschaulich:

- Jede Nebenbedingung der Form

$$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i$$

  beschreibt einen *Halbraum* im $\mathbb{R}^n$.
- Da *alle* Nebenbedingung erfüllt sein müssen definiert der *Schnitt* der Nebenbedingungen die Menge aller Lösungen – ein *konvexer Polyeder*.
- Der Vektor c zeigt in die zu optimierende Richtung.

Simplex-Algorithmus (Geometrisch)

Das Vorgehen anschaulich:

- Jede Nebenbedingung der Form

$$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i$$

  beschreibt einen *Halbraum* im $\mathbb{R}^n$.
- Da *alle* Nebenbedingung erfüllt sein müssen definiert der *Schnitt* der Nebenbedingungen die Menge aller Lösungen – ein *konvexer Polyeder*.
- Der Vektor c zeigt in die zu optimierende Richtung.

Da der Lösungspolyeder konvex ist, ist die optimale Lösung stets eine *Ecke* des Polyeders.

Simplex-Algorithmus (Geometrisch)

Das Vorgehen anschaulich:

- Jede Nebenbedingung der Form

$$a_{i1}x_1 + \ldots + a_{in}x_n \leq b_i$$

  beschreibt einen *Halbraum* im $\mathbb{R}^n$.
- Da *alle* Nebenbedingung erfüllt sein müssen definiert der *Schnitt* der Nebenbedingungen die Menge aller Lösungen – ein *konvexer Polyeder*.
- Der Vektor c zeigt in die zu optimierende Richtung.

Da der Lösungspolyeder konvex ist, ist die optimale Lösung stets eine *Ecke* des Polyeders.

- Wir bewegen uns von Ecke zu Ecke entlang „verbessernder“ Kanten.

Simplex-Algorithmus (Geometrisch)

Algorithmus 4 : SimplexGeom

Eingabe : Lineares Programm (A, c, b) in Standardform
Ausgabe : Optimale Lösung x des LPs

1 $P \leftarrow$ konvexes Lösungspolyeder zu (A, c, b)
2 $x \leftarrow$ beliebige Ecke in P
3 **solange** *es gibt verbessernde Kante* $(x, y) \in P$ **tue**
4 $\quad x \leftarrow y$
5 return x

$\rightsquigarrow$ Wir bewegen uns über die Kanten des Polyeders von Ecke zu Ecke, bis wir eine optimale Ecke x gefunden haben.

Simplex-Algorithmus (Algebraisch)

Der „Ecken-Tausch“ lässt sich auch algebraisch interpretieren.

Simplex-Algorithmus (Algebraisch)

Der „Ecken-Tausch“ lässt sich auch algebraisch interpretieren.

- Statt sich von Ecke zu Ecke zu bewegen transformieren wir das Koordinatensystem:

Simplex-Algorithmus (Algebraisch)

Der „Ecken-Tausch“ lässt sich auch algebraisch interpretieren.

- Statt sich von Ecke zu Ecke zu bewegen transformieren wir das Koordinatensystem:
  - Der Nullpunkt ist stets die aktuell zu betrachtende Ecke.

Simplex-Algorithmus (Algebraisch)

Der „Ecken-Tausch“ lässt sich auch algebraisch interpretieren.

- Statt sich von Ecke zu Ecke zu bewegen transformieren wir das Koordinatensystem:
  - Der Nullpunkt ist stets die aktuell zu betrachtende Ecke.
  - Die Koordinatenachsen laufen entlang der zur Ecke inzidenten Kanten.

Simplex-Algorithmus (Algebraisch)

Der „Ecken-Tausch“ lässt sich auch algebraisch interpretieren.

- Statt sich von Ecke zu Ecke zu bewegen transformieren wir das Koordinatensystem:
  - Der Nullpunkt ist stets die aktuell zu betrachtende Ecke.
  - Die Koordinatenachsen laufen entlang der zur Ecke inzidenten Kanten.
- Zentrale Operation: Eckentausch $\leftrightarrow$ Koordinatentransformation.

Simplex-Algorithmus (Algebraisch)

Der „Ecken-Tausch“ lässt sich auch algebraisch interpretieren.

- Statt sich von Ecke zu Ecke zu bewegen transformieren wir das Koordinatensystem:
  - Der Nullpunkt ist stets die aktuell zu betrachtende Ecke.
  - Die Koordinatenachsen laufen entlang der zur Ecke inzidenten Kanten.
- Zentrale Operation: Eckentausch $\leftrightarrow$ Koordinatentransformation.
- Der Nullpunkt muss zu Anfang bereits eine Ecke des Polyeders sein!

Simplex (Algebraisch)

Algorithmus 5 : SimplexAlg

Eingabe : Lineares Programm (B, b, c, v) in Normalform
Ausgabe : Optimale Lösung des LPs

1 $y \leftarrow$ Basislösung.

2 **wenn** $\exists$ *Variable* x_j *in ZF mit positivem Koeffizienten* $c_j > 0$ **dann**

3 Erhöhe x_j maximal. Finde also Ungleichung i, die das Erhöhen von x_j am stärksten einschränkt.

4 Tausche x_j mit y_i. Forme dazu Gleichung i nach x_j um, und setze diese für alle Vorkommen von x_j ein (auch in ZF).

5 Weiter in Zeile 2

6 **sonst**

7 $(x_1, \ldots, x_m) = 0$ ist beste Lösung.

Aufgabe

Simplex-Aufgabe

Gegeben sei folgendes lineares Programm.

$$\textbf{maximiere } 4x_1 + x_2$$

unter

$$\begin{array}{rcrcl} x_1 & + & x_2 & \leq & 16 \\ & & x_2 & \leq & 12 \\ 3x_1 & + & x_2 & \leq & 36 \end{array}$$

(a) Lösen Sie das LP geometrisch.

(b) Lösen Sie das LP algebraisch.

Frage

Könnt ihr nochmal einen Überblick über die verschiedenen Optimierungsverfahren geben?

Definitionen

Optimierungsproblem

Optimierungsproblem $\mathcal{P} = (Q, c, I)$ bestehend aus

- Suchraum Q
- Bewertungs-(Kosten-)funktion c, die jedem Zustand $q \in Q$ seine Kosten $c(q)$ zuordnet.
- Straffunktion I, die in Q durch $I(q) = 0$ Lösungen charakterisiert
- ist *kombinatorisch*, wenn der Suchraum Q diskret ist
- $c'(q) = c(q) + \gamma I(q)$

Schrittweise Optimierung

Gradientenverfahren

- Zustandsraum $Q \subset \mathbb{R}^n$, jeder Zustand durch Punkt $q = (x_1, \ldots, x_n)$ repräsentiert.
- Ausgehend von Punkt $q = q_0$ wird der Gradient

$$\nabla c(q) = grad\ c(q) = (\frac{\partial c}{\partial x_1}, \ldots, \frac{\partial c}{\partial x_n})$$

  der Kostenfunktion c an Stelle q bestimmt.
- Nachfolger q' wird definiert durch $q' = (x_1 \pm h \cdot \frac{\partial c}{\partial x_1}, \ldots, x_n \pm h \cdot \frac{\partial c}{\partial x_m})$
- Schrittweite h bestimmt, wie weit man in Richtung $\nabla c(q)$ folgt.

Schrittweise Optimierung

Bewertung - Gradientenverfahren

- deterministisch
- kann nicht aus lok. Opt. entkommen
- nicht anwendbar, wenn:
  - Kostenfunktion nicht stetig diff'bar
  - Ableitungsberechnung hat hohen Aufwand
  - lokale Optima liegen auf dem Weg zum globalen Optimum

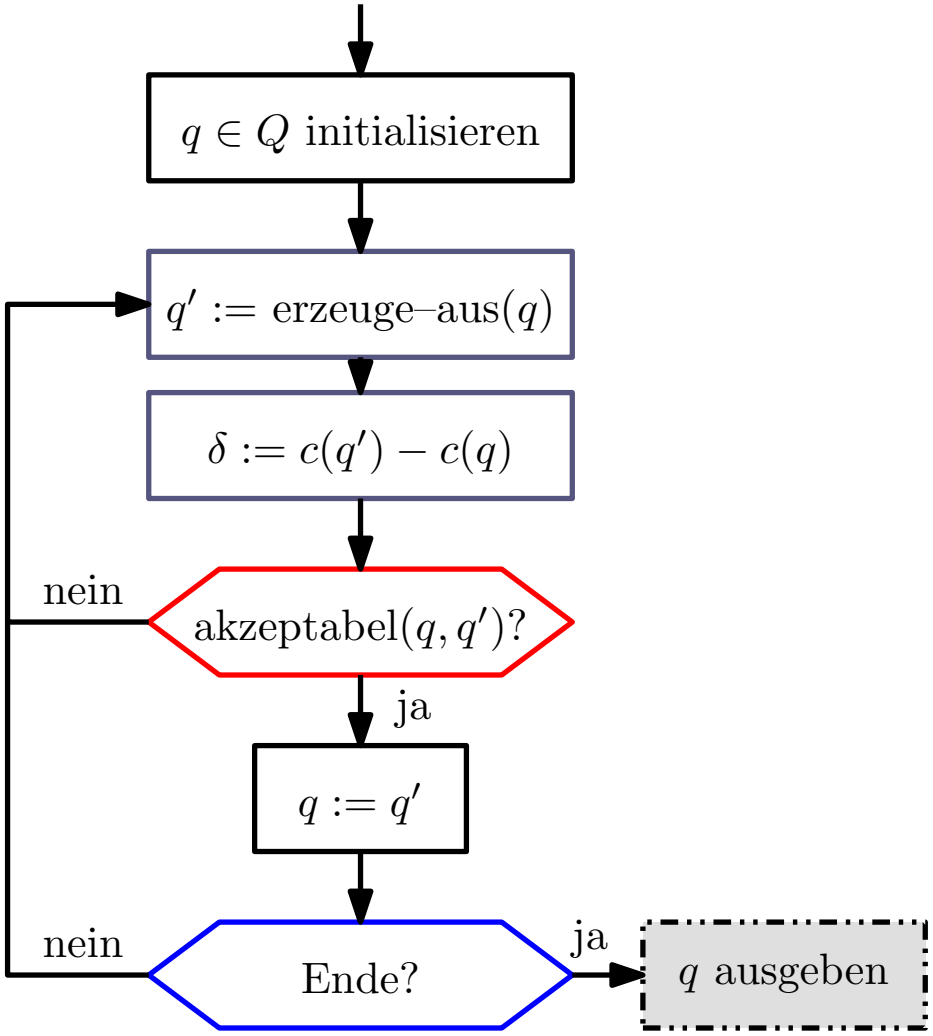
Gradientenverfahren - Beispiel

Aufgabe

Gegeben sei $f(x,y) := sin(x) * cos(y)$ sowie die Startlösung $(x,y) = (\frac{\pi}{6}, \frac{\pi}{3})$. Finden sie mittels Gradientenverfahren ein Minumum.
Hinweis: Ihnen könnte folgende Wertetabelle nützlich erscheinen:

	0	$\frac{\pi}{6}$	$\frac{\pi}{4}$	$\frac{\pi}{3}$	$\frac{\pi}{2}$
$sin(x)$	$\frac{1}{2}\sqrt{0}$	$\frac{1}{2}\sqrt{1}$	$\frac{1}{2}\sqrt{2}$	$\frac{1}{2}\sqrt{3}$	$\frac{1}{2}\sqrt{4}$
$cos(x)$	$\frac{1}{2}\sqrt{4}$	$\frac{1}{2}\sqrt{3}$	$\frac{1}{2}\sqrt{2}$	$\frac{1}{2}\sqrt{1}$	$\frac{1}{2}\sqrt{0}$

allgemeiner stochastischer Optimierungsalgorithmus



Simuliertes Tempern (Eigenschaften)

- im heißen Zustand Wahrscheinlichkeit 1, dass schlechtere Lösung akzeptiert werden
- erreicht Optimum beweisbar, wenn T unendlich langsam gesenkt wird
- Tradeoff zwischen Rechenzeit und Genauigkeit
- Schwierigkeit in Wahl der Parameter
- Accept: $\rightarrow c(p) < c(q)$ or $e^{\frac{c(q)-c(p)}{t_i}} > Random\{0,1\}$
- Updates: T sinkt
- kann aus lokalen Minima entkommen (Hill Climbing)

Schwellwert-Algorithmus (threshold accepting)

- deterministische Akzeptanzbedingung
- Ein mutierter Wert q' wird akzeptiert, wenn seine Bewertung die des bisherigen Wertes höchstens um einen Schwellwert S überschreitet
- Initialisierung: $S :=$ Schwellwert $S > 0$
- Akzeptanzbedingung: $accept \rightarrow c(q') \leq c(q) + S$
- Updates: S wird regelmäßig gesenkt, sobald $S = 0$ gibt es keine Rückschritte mehr
- kann aus lokalen Minima entkommen (Hill Climbing)

Sintflut-Algorithmus (great deluge algorithm)

- obere Grenze H, die nicht überschritten werden darf, sinkt immer weiter ab, z.B. $H = 1.5q_0$
- Ein mutierter Wert q' wird akzeptiert, wenn seine Bewertung die Sintflutgrenze unterschreitet (hängt also nicht vom Zustand q ab)
- Initialisierung $H = 1.5q_0$
- Akzeptanzbedingung: $accept \rightarrow c(q') \leq H$
- Updates: Die Sintflut H wird regelmäßig gesenkt
- gefangen in lokalen Minima

Rekordjagd (record to record travel)

- bisher beste (geringste) Bewertung (*record*) wird gespeichert
- bisheriger Rekord darf nur um einen bestimmten Wert Δ überschritten werden. Ein neuer *record* wird automatisch gespeichert.
- erzeugter Wert q' wird akzeptiert, wenn seine Bewertung den bisherigen Rekord um einen Maximalwert Δ überschreitet
- Initialisierung: record $:= c(q_o)$
- Akzeptanzbedingung: $accept \rightarrow c(q') \leq record + \Delta$
- Updates: *record* wird bei einem neuen besten Wert automatisch aktualisiert, Δ wird regelmäßig abgesenkt.
- gefangen in lokalen Minima

Evolutionäre und Genetische Algorithmen (Bewertung)

- Mischen/Mitteln als primäre Suchstrategie (EA)
- Kreuzung als primäre Suchstrategie (GA)
- Schwierigkeit liegt insb. in der Codierung des Problems, d.h. "Was ist *kreuzen* auf zwei Offset-Assignments?"
- Keine mathematische Grundlage, keine Aussagen, wie gut EA/GA optimieren
- EA konvergiert i.A. schneller, aber Gefahr im lok. Opt. zu enden erhöht sich im gleichen Maße
- GAs spähen wegen der Kreuzung den Suchraum in größeren Sprüngen aus, steigern damit Chance, auf glob. Optimum zu stoßen, aber sehr schlechte Konvergenzeigenschaften

Frage

Könntet ihr nochmal kurz Backpropagation für mehrschichtige Neuronale Netze wiederholen?

Künstliche Intelligenz ist allemal besser als natürliche Dummheit.

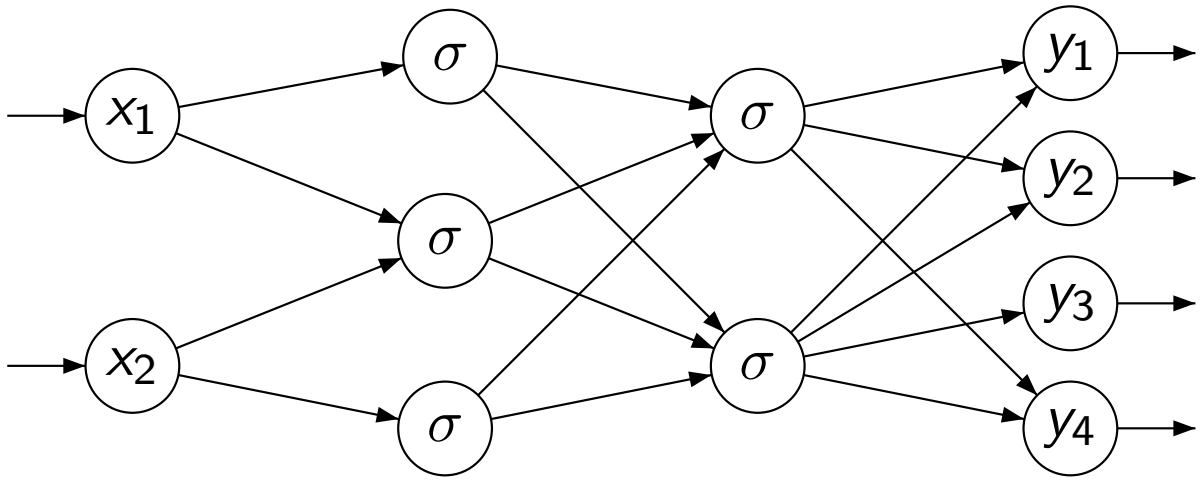
Hans Matthöfer (*1925)

Wiederholung: Perzeptron

Perzeptron

Das *Perzeptron* ist ein vorwärtsgerichtetes Neuronales Netz, bei dem die Neuronen schichtweise angeordnet sind. Verbindungen existieren nur zwischen benachbarten Schichten.

Das *einfache Perzeptron* besitzt nur eine Eingabe- sowie eine Ausgabeschicht.



Wiederholung: Neuronale Netze

Arbeitsweise der Neuronen

Die Neuronen arbeiten alle *synchron* in diskreten Zeitabständen. Jedes Neuron N_j hat einen *Zustand* (bzw. eine Ausgabe) $q_j \in Q$, der an den Ausgabekanten anliegt.

Die *Erregungsfunktion* p_j des Neurons N_j berechnet sich aus den Werten der Eingangskanten sowie dem Schwellwert von N_j:

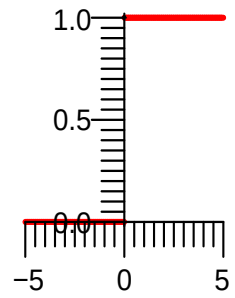
$$p_j := \sum_i w_{ij} q_i - \sigma_j$$

Die *Aktivierungsfunktion* $h_j(p_j)$ berechnet die *Ausgabe* (Zustand) des Neurons, und ist i.A. eine *sigmoide Funktion*. Es ist $q_j := h_j(p_j)$

Wiederholung: Neuronale Netze; Aktivierungsfunktionen

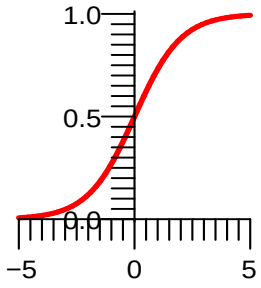
Zustandsmenge $Q = \{0, 1\}$ und

$$h(p) := \begin{cases} 1 & \text{falls} \quad p \geq 0 \\ 0 & \text{sonst} \end{cases}$$



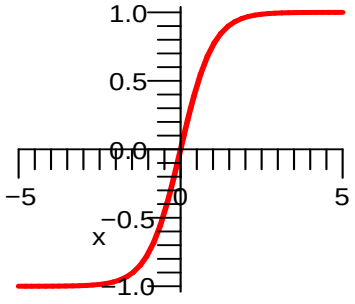
Zustandsmenge $Q = [0, 1]$ und

$$h(p) := \frac{1}{1 + e^{-\beta p}}$$



Zustandsmenge $Q = [-1, 1]$ und

$$h(p) := \tanh(\beta p)$$



Backpropagation-Algorithmus

Das Problem

Gegeben: Eine Menge von Trainingsdaten und zugehöriger Klassifikation (Ausgabe).

Gesucht: Gewichte und Schwellwerte, so dass das Netz die Trainingsdaten gut klassifiziert.

Backpropagation-Algorithmus

Das Problem

Gegeben: Eine Menge von Trainingsdaten und zugehöriger Klassifikation (Ausgabe).

Gesucht: Gewichte und Schwellwerte, so dass das Netz die Trainingsdaten gut klassifiziert.

$\rightsquigarrow$ Die gewünschte Ausgabe der versteckten Neuronen ist unbekannt!

Backpropagation-Algorithmus

Das Problem

Gegeben: Eine Menge von Trainingsdaten und zugehöriger Klassifikation (Ausgabe).

Gesucht: Gewichte und Schwellwerte, so dass das Netz die Trainingsdaten gut klassifiziert.

$\rightsquigarrow$ Die gewünschte Ausgabe der versteckten Neuronen ist unbekannt!

Idee des PP-Algorithmus

Definiere Fehlerfunktion $E(w)$ die von den Gewichten abhängt, und minimiere diese mittels *Gradientenabstieg*.

Backpropagation-Algorithmus

Das Vorgehen anschaulich:

1. Forward-Pass (Schichtweises Durchreichen der Eingabe)
2. Bestimmung des Fehlers
3. Backpropagation (Rückrechnen des Fehlers)
4. Modifizieren der Gewichte
5. Goto (1)

Backpropagation-Algorithmus

Das Vorgehen anschaulich:

1. Forward-Pass (Schichtweises Durchreichen der Eingabe)
2. Bestimmung des Fehlers
3. Backpropagation (Rückrechnen des Fehlers)
4. Modifizieren der Gewichte
5. Goto (1)

Zwei Methoden:

- Musterlernen: Fehler wird für *ein* Trainingsdatum errechnet
- Epochelernen: Fehler wird für *alle* Trainingsdaten errechnet

Backpropagation-Algorithmus

Das Vorgehen anschaulich:

1. Forward-Pass (Schichtweises Durchreichen der Eingabe)
2. Bestimmung des Fehlers
3. Backpropagation (Rückrechnen des Fehlers)
4. Modifizieren der Gewichte
5. Goto (1)

Zwei Methoden:

- Musterlernen: Fehler wird für *ein* Trainingsdatum errechnet
- Epochelernen: Fehler wird für *alle* Trainingsdaten errechnet

Backpropagation-Algorithmus

Sei $x_1, \ldots, x_M$ eine Menge von Trainingseingaben mit den korrekten Ausgaben $a_1, \ldots, a_M$. Vermögen weiterhin W die Gewichte aller Synapsen.

Backpropagation-Algorithmus

Sei $x_1, \ldots, x_M$ eine Menge von Trainingseingaben mit den korrekten Ausgaben $a_1, \ldots, a_M$. Vermögen weiterhin W die Gewichte aller Synapsen.

- Die zu minimierende Fehlerfunktion ist

$$E(W) = \frac{1}{2}\|a - q\|^2 = \frac{1}{2}\sum_j (a_j - q_j)^2$$

Backpropagation-Algorithmus

Sei $x_1, \ldots, x_M$ eine Menge von Trainingseingaben mit den korrekten Ausgaben $a_1, \ldots, a_M$. Vermögen weiterhin W die Gewichte aller Synapsen.

- Die zu minimierende Fehlerfunktion ist

$$E(W) = \frac{1}{2}\|a - q\|^2 = \frac{1}{2}\sum_j (a_j - q_j)^2$$

- Benutze Gradientenabstieg. Also Rechenvorschrift:

$$W^+ = W - \eta \cdot \nabla E(W)$$

  Bzw. aufgeschlüsselt nach Gewichten:

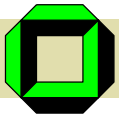
$$w_{ij}^+ = w_{ij} \underbrace{-\eta \cdot \frac{\partial E}{\partial w_{ij}}}_{=:\Delta w_{ij}}$$

Backpropagation-Algorithmus

Mit recht viel Rechnerei (Mehrfache Anwendung der Kettenregel[1]) folgt

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot q_i$$

[1] Siehe Tutorium vom 27. Mai 2007



Backpropagation-Algorithmus

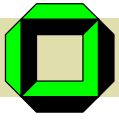
Mit recht viel Rechnerei (Mehrfache Anwendung der Kettenregel[1]) folgt

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot q_i$$

Dabei sind

- η – Schrittweite beim Gradientenabstieg (frei wählbar)
- q_i – Zustand des betrachteten Neurons in Schicht i
- δ_j – „Fehlerkoeffizient“ aus Schicht $j = i + 1$

[1] Siehe Tutorium vom 27. Mai 2007



Backpropagation-Algorithmus

Mit recht viel Rechnerei (Mehrfache Anwendung der Kettenregel[1]) folgt

$$\Delta w_{ij} = \eta \cdot \delta_j \cdot q_i$$

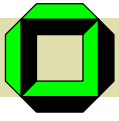
Dabei sind

- η – Schrittweite beim Gradientenabstieg (frei wählbar)
- q_i – Zustand des betrachteten Neurons in Schicht i
- δ_j – „Fehlerkoeffizient“ aus Schicht $j = i + 1$

Berechnung von δ_j nach folgender Regel:

$$\delta_j := \begin{cases} h'(p_j)(a_j - q_j) & \text{falls } N_j \text{ Ausgabe-Neuron} \\ h'(p_j) \sum_k \delta_k w_{jk} & \text{falls } N_j \text{ Hidden-Neuron} \end{cases}$$

[1] Siehe Tutorium vom 27. Mai 2007



Backpropagation-Algorithmus

Hinweise zum Backpropagation Algorithmus.

- Die Werte p_j und $q_j = h(p_j)$ sollten im Vorwärtsschritt zwischengespeichert werden.

Backpropagation-Algorithmus

Hinweise zum Backpropagation Algorithmus.

- Die Werte p_j und $q_j = h(p_j)$ sollten im Vorwärtsschritt zwischengespeichert werden.
- Gewichte nicht mit 0 initialisieren. $\rightsquigarrow$ Zufall.

Backpropagation-Algorithmus

Hinweise zum Backpropagation Algorithmus.

- Die Werte p_j und $q_j = h(p_j)$ sollten im Vorwärtsschritt zwischengespeichert werden.
- Gewichte nicht mit 0 initialisieren. $\rightsquigarrow$ Zufall.
- Alle Probleme des Gradientenabstiegs treffen hier auch zu.

Backpropagation-Algorithmus

Hinweise zum Backpropagation Algorithmus.

- Die Werte p_j und $q_j = h(p_j)$ sollten im Vorwärtsschritt zwischengespeichert werden.
- Gewichte nicht mit 0 initialisieren. $\rightsquigarrow$ Zufall.
- Alle Probleme des Gradientenabstiegs treffen hier auch zu.
- Netztopologie muss „manuell“ im Voraus festgelegt werden. $\rightsquigarrow$ Problem der Generalisierungsfähigkeit.

Das Ende Ist Nahe. . .

Ein paar Wahr/Falsch Fragen.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$. X

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$. X
- Ein Hopfield-Netz erreicht immer nach endlich vielen Schritten einen stabilen Zustand.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$. X
- Ein Hopfield-Netz erreicht immer nach endlich vielen Schritten einen stabilen Zustand. ✓

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$. X
- Ein Hopfield-Netz erreicht immer nach endlich vielen Schritten einen stabilen Zustand. ✓
- Ein Kanal ist deterministisch falls $H(Y|X) = 0$.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$. X
- Ein Hopfield-Netz erreicht immer nach endlich vielen Schritten einen stabilen Zustand. ✓
- Ein Kanal ist deterministisch falls $H(Y|X) = 0$. ✓

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$. X
- Ein Hopfield-Netz erreicht immer nach endlich vielen Schritten einen stabilen Zustand. ✓
- Ein Kanal ist deterministisch falls $H(Y|X) = 0$. ✓
- Der Kullback-Leibler-Abstand $D(p\|q)$ ist eine Metrik.

Wahr/Falsch-Fragen

- Bei evolutionären Algorithmen können die Eltern im Gegensatz zu den genetischen Algorithmen immer überleben. X
- Es gibt keinen Code C so dass $L(C) < L(S)$ wobei S der Shannoncode ist. X
- In einem mehrschichtigen Perzeptron reicht eine verdeckte Schicht nicht aus, um jede berechenbare stetige Funktion beliebig gut anzunähern. X
- Es gibt Voronoi-Gebiete G für die gilt: $\lambda a + (1 - \lambda)b \notin G$ für $a, b \in G$ und $0 \leq \lambda \leq 1$. X
- Ein Hopfield-Netz erreicht immer nach endlich vielen Schritten einen stabilen Zustand. ✓
- Ein Kanal ist deterministisch falls $H(Y|X) = 0$. ✓
- Der Kullback-Leibler-Abstand $D(p\|q)$ ist eine Metrik. X

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben.

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer.

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$.

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar.

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar. X

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar. X
- Bei der Übertragung von LZW kodierten Daten braucht das Wörterbuch oft den meisten Platz.

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar. X
- Bei der Übertragung von LZW kodierten Daten braucht das Wörterbuch oft den meisten Platz. X

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar. X
- Bei der Übertragung von LZW kodierten Daten braucht das Wörterbuch oft den meisten Platz. X
- Bei der Wavelettransformation findet kein Informationsverlust statt.

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar. X
- Bei der Übertragung von LZW kodierten Daten braucht das Wörterbuch oft den meisten Platz. X
- Bei der Wavelettransformation findet kein Informationsverlust statt. ✓

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar. X
- Bei der Übertragung von LZW kodierten Daten braucht das Wörterbuch oft den meisten Platz. X
- Bei der Wavelettransformation findet kein Informationsverlust statt. ✓
- Nach Dueck, Scheuer und Wallmeister ist das Simulierte Tempern dem Sintflut- und Schwellwert-Verfahren überlegen.

Wahr/Falsch-Fragen

- In einem rekurrenten Neuronalen Netz kann es Rückwärtskanten geben. ✓
- Der Lernalgorithmus für das einfache Perzeptron terminiert immer. X
- Haben die Codeworte x_i mindestens den Abstand

$$d(x_i, x_j) \geq 2e + 1 \qquad \text{für alle } i \neq j$$

können f falsche Bits korrigiert werden, wenn $f \leq e$. ✓
- Reguläre Kodes sind stets dekodierbar. X
- Bei der Übertragung von LZW kodierten Daten braucht das Wörterbuch oft den meisten Platz. X
- Bei der Wavelettransformation findet kein Informationsverlust statt. ✓
- Nach Dueck, Scheuer und Wallmeister ist das Simulierte Tempern dem Sintflut- und Schwellwert-Verfahren überlegen. X

Das Ende

Noch Fragen?

Das Ende

Don't Panic!

Das Ende

Viel Erfolg in der Klausur und im weiteren Studium!