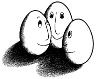
Complementing search engines with online web mining agents

Filippo Menczer (2003)

Sebastian Land

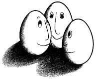
TU Dortmund

20.11.2007 Seminar “Intelligente Anwendungen im Internet”



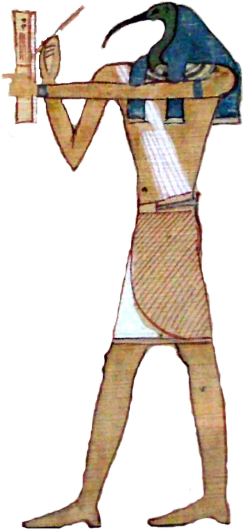
Outline

Outline



Altertum

- Informationen liegen in handgeschriebenen Schriften vor
- Retrieval durch Lesen

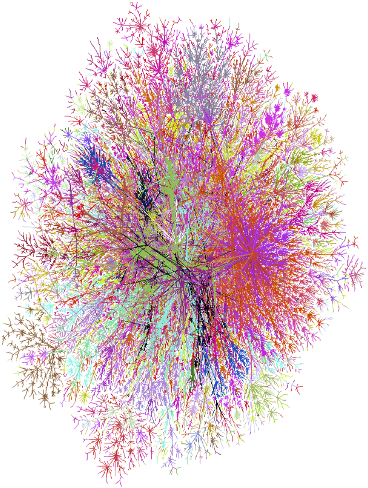


Neuzeit

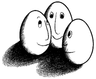
- Einführung des Buchdrucks erleichtert Vervielfältigung von Schriften
- Entstehung großer Bibliotheken und erster Verzeichnisse

Informationszeitalter

- Einführung von Computern und Internet vermehrt(e) zugängliche Schriften explosionsartig
- Retrieval durch Lesen unmöglich
- Daher manuelle Erstellung von Verzeichnissen sehr aufwändig
- Schriften nicht mehr statisch: Können Qualität und Inhalt täglich ändern
- Dies führt statische Verzeichnisse ad absurdum



Outline



"Information Retrieval"

Information Retrieval ist die Wissenschaft, die sich mit computergestütztem inhaltsorientiertem Suchen beschäftigt. [. . .] Wie der Begriff *Retrieval* (deutsch *Wiedergewinnung, Auffindung*) sagt, sind Informationen in großen Datenbeständen zunächst verloren und müssen wieder gewonnen bzw. wieder gefunden werden.

Information Retrieval stellt verschiedene Werkzeuge zur Verfügung:

- Term Frequency
- Cosinus Ähnlichkeit
- Recall und Precision

Term Frequency

Die Term Frequency (TF) ist ein Vektor der Häufigkeiten aller Worte in einem Dokument. Sinnvollerweise werden Stopworte entfernt und die Worte in ihre Grundform überführt.

Beispiel:

Original	Die Term Frequency ist ein Vektor der Frequencies aller Terme in einem Dokument			
ohne Stopworte	Term Frequency Vektor Frequencies Terme Dokument			
Grundformen	Term Frequency Vektor Frequency Term Dokument			
Vektor	2 Term	2 Frequency	1 Vektor	1 Dokument

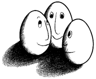
Cosinus Ähnlichkeit

Die Cosinus Ähnlichkeit misst Ähnlichkeit θ zwischen zwei Dokumenten. Dazu benutzt sie den Winkel zwischen den TF-Vektoren der beiden Dokumente:

$$\begin{aligned}\theta &= \angle(A, B) \\ &= \arccos \frac{A \cdot B}{|A| * |B|} \\ &= \arccos \frac{\sum_i a_i \cdot b_i}{\sqrt{\sum_i a_i^2 \cdot \sum_i b_i^2}}\end{aligned}$$

Nachteile:

- Für alle Dokumente muss Term Frequency bekannt sein
- Berechnung bei vielen Dokumenten aufwendig



Beispiel

$$\vec{q} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \qquad \vec{d} = \begin{pmatrix} 2 \\ 0 \\ 3 \\ 1 \end{pmatrix}$$

$$\begin{aligned}
\theta &= \arccos\left(\frac{1\cdot 2 + 1\cdot 0 + 1\cdot 3 + 1\cdot 1}{\sqrt{(1^2+1^2+1^2+1^2)\cdot(2^2+3^2+0^2+1^2)}}\right) \\
&= \arccos\left(\frac{6}{\sqrt{4\cdot 14}}\right) \\
&= \arccos(0.80178) \\
&= 0.64
\end{aligned}$$

Recall

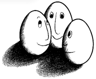
Der Recall ist ein Gütemaß für Suchergebnisse. Er gibt den Anteil der gefundenen relevanten an der Gesamtmenge der relevanten Dokumente an:

$$\text{Recall} = \frac{|\{\text{relevante Dokumente}\} \cap \{\text{gefundene Dokumente}\}|}{|\{\text{relevante Dokumente}\}|}$$

Precision

Die Precision ist ein Gütemaß für Suchergebnisse. Sie gibt den Anteil der gefundenen relevanten an der Menge der gefundenen Dokumente an:

$$\text{Precision} = \frac{|\{\text{relevante Dokumente}\} \cap \{\text{gefundene Dokumente}\}|}{|\{\text{gefundene Dokumente}\}|}$$



Recall

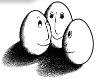
Der Recall ist ein Gütemaß für Suchergebnisse. Er gibt den Anteil der gefundenen relevanten an der Gesamtmenge der relevanten Dokumente an:

$$\text{Recall} = \frac{|\{\text{relevante Dokumente}\} \cap \{\text{gefundene Dokumente}\}|}{|\{\text{relevante Dokumente}\}|}$$

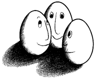
Precision

Die Precision ist ein Gütemaß für Suchergebnisse. Sie gibt den Anteil der gefundenen relevanten an der Menge der gefundenen Dokumente an:

$$\text{Precision} = \frac{|\{\text{relevante Dokumente}\} \cap \{\text{gefundene Dokumente}\}|}{|\{\text{gefundene Dokumente}\}|}$$



Outline



Indexsuchmaschinen

Indexsuchmaschinen suchen auf Wortlisten, den sogenannten Indexlisten. Diese werden vor der Suche erstellt und nehmen gefundene Worte und deren Fundstellen auf. Aktuelle Suchmaschinen benutzen Crawler, die die Internetseiten regelmäßig besuchen um Änderungen aufzunehmen.

Nachteile:

- Durchsucht niemals “Jetzt”-Zustand des Internets: Mangelnde Aktualität
- Gefunden werden nur Dokumente, die die Suchwörter enthalten

Outline

Künstliche neuronale Netze

- Lernverfahren, inspiriert von menschlicher Art des Lernens
- Basieren dabei auf mehr oder weniger komplexen Graphen
- Trainierte Modelle lassen sich entsprechend schlecht interpretieren
- In der Hypezeit wurden sie daher fast als magisch behandelt
- Heute entzaubert: “Nur” noch nichtlineare statistische Modelle

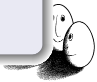
Vergleich menschliches und künstliches neuronales Netz

Das Gehirn

- besitzt ca. 10^{11} Neuronen
- jedes Neuron hat ca. 10^4 Verbindungen
- Schaltzeit eines Neurons ca 10^{-3} Sekunden
- Entscheidung in $0,1$ Sekunden
- entsprechend bis zu 100 Hops

Künstliches neuronales Netz

- wenige hundert bis einige tausend Neuronen
- Anzahl Verbindungen in selber Größenordnung
- Schaltzeit eines Computers ca 10^{-10} Sekunden
- Entscheidung in Sekundenbruchteilen, aber:
- nur sehr wenige Hops, selten mehr als 4



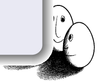
Vergleich menschliches und künstliches neuronales Netz

Das Gehirn

- besitzt ca. 10^{11} Neuronen
- jedes Neuron hat ca. 10^{4} Verbindungen
- Schaltzeit eines Neurons ca 10^{-3} Sekunden
- Entscheidung in $0,1$ Sekunden
- entsprechend bis zu 100 Hops

Künstliches neuronales Netz

- wenige hundert bis einige tausend Neuronen
- Anzahl Verbindungen in selber Größenordnung
- Schaltzeit eines Computers ca 10^{-10} Sekunden
- Entscheidung in Sekundenbruchteilen, aber:
- nur sehr wenige Hops, selten mehr als 4



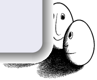
Vergleich menschliches und künstliches neuronales Netz

Das Gehirn

- besitzt ca. 10^{11} Neuronen
- jedes Neuron hat ca. 10^4 Verbindungen
- Schaltzeit eines Neurons ca 10^{-3} Sekunden
- Entscheidung in $0,1$ Sekunden
- entsprechend bis zu 100 Hops

Künstliches neuronales Netz

- wenige hundert bis einige tausend Neuronen
- Anzahl Verbindungen in selber Größenordnung
- Schaltzeit eines Computers ca 10^{-10} Sekunden
- Entscheidung in Sekundenbruchteilen, aber:
- nur sehr wenige Hops, selten mehr als 4



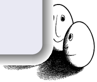
Vergleich menschliches und künstliches neuronales Netz

Das Gehirn

- besitzt ca. 10^{11} Neuronen
- jedes Neuron hat ca. 10^4 Verbindungen
- Schaltzeit eines Neurons ca 10^{-3} Sekunden
- Entscheidung in $0,1$ Sekunden
- entsprechend bis zu 100 Hops

Künstliches neuronales Netz

- wenige hundert bis einige tausend Neuronen
- Anzahl Verbindungen in selber Größenordnung
- Schaltzeit eines Computers ca 10^{-10} Sekunden
- Entscheidung in Sekundenbruchteilen, aber:
- nur sehr wenige Hops, selten mehr als 4



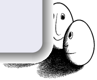
Vergleich menschliches und künstliches neuronales Netz

Das Gehirn

- besitzt ca. 10^{11} Neuronen
- jedes Neuron hat ca. 10^4 Verbindungen
- Schaltzeit eines Neurons ca 10^{-3} Sekunden
- Entscheidung in $0,1$ Sekunden
- entsprechend bis zu 100 Hops

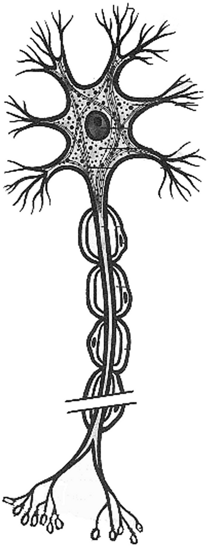
Künstliches neuronales Netz

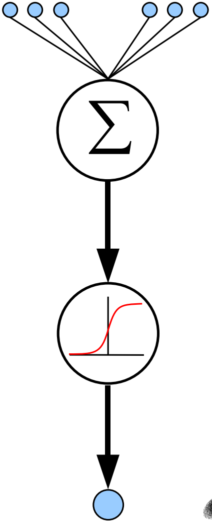
- wenige hundert bis einige tausend Neuronen
- Anzahl Verbindungen in selber Größenordnung
- Schaltzeit eines Computers ca 10^{-10} Sekunden
- Entscheidung in Sekundenbruchteilen, aber:
- nur sehr wenige Hops, selten mehr als 4

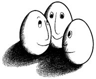


Vergleich menschliches und künstliches Neuron

- Analogien offensichtlich
- **Aber:** Unterschiede in Arbeitsweise: analoge Zeitreihen vs. transformierte Summe







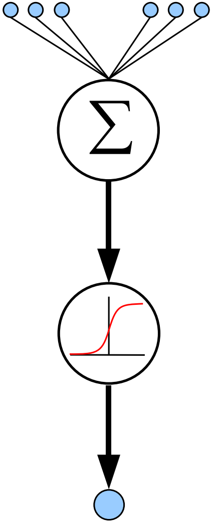
Arbeitsweise

- Eingabewerte werden mit Gewichten w_i oder $\omega_{i,j}$ multipliziert
- Darüber wird die Summe gebildet
- Diese Summe wird mit einer Aktivierungsfunktion σ transformiert
- Der Wert der Aktivierungsfunktion wird ausgegeben

Verschiedene Aktivierungsfunktionen möglich:

- Sigmoid-Funktion $\sigma(x) = \frac{1}{1+e^{-x}}$
- Identität $\sigma(x) = x$
- Signum $\sigma(x) = sign(x)$

Praktisch relevant jedoch nur differenzierbare Funktionen!

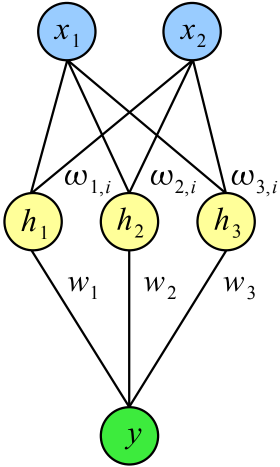


Einfaches Künstliches neuronales Netz

Bestandteile:

- 2 Inputs x_1, x_2
- 3 Neuronen h_j im hidden Layer: h_1, h_2, h_3
- mit je 2 Gewichten $\omega_{j,1}, \omega_{j,2}$
- und einem Ziel y mit Gewichten w_1, w_2, w_3

Generell sind mehr hidden Layer, mehr Neuronen in den hidden Layern, aber auch mehr als eine Zielvariable möglich!

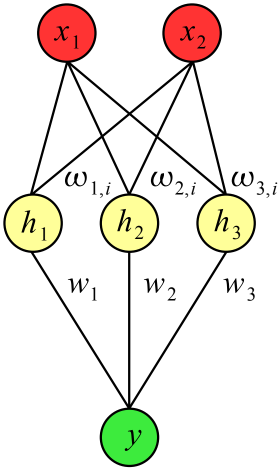


Einfaches Künstliches neuronales Netz

Bestandteile:

- 2 Inputs x_1, x_2
- 3 Neuronen h_j im hidden Layer: h_1, h_2, h_3
- mit je 2 Gewichten $\omega_{j,1}, \omega_{j,2}$
- und einem Ziel y mit Gewichten w_1, w_2, w_3

Generell sind mehr hidden Layer, mehr Neuronen in den hidden Layern, aber auch mehr als eine Zielvariable möglich!

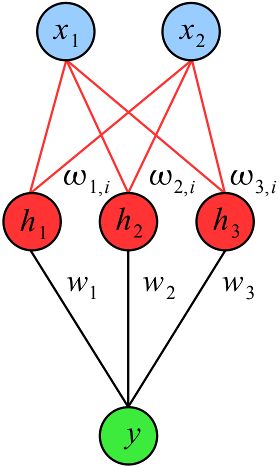


Einfaches Künstliches neuronales Netz

Bestandteile:

- 2 Inputs x_1, x_2
- 3 Neuronen h_j im hidden Layer: h_1, h_2, h_3
- mit je 2 Gewichten $\omega_{j,1}, \omega_{j,2}$
- und einem Ziel y mit Gewichten w_1, w_2, w_3

Generell sind mehr hidden Layer, mehr Neuronen in den hidden Layern, aber auch mehr als eine Zielvariable möglich!

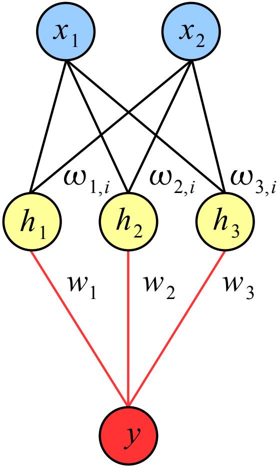


Einfaches Künstliches neuronales Netz

Bestandteile:

- 2 Inputs x_1, x_2
- 3 Neuronen h_j im hidden Layer: h_1, h_2, h_3
- mit je 2 Gewichten $\omega_{j,1}, \omega_{j,2}$
- und einem Ziel y mit Gewichten w_1, w_2, w_3

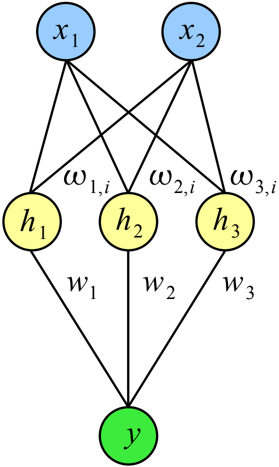
Generell sind mehr hidden Layer, mehr Neuronen in den hidden Layern, aber auch mehr als eine Zielvariable möglich!



Funktionsweise

Vorgehen um neuronales Netz anzuwenden:

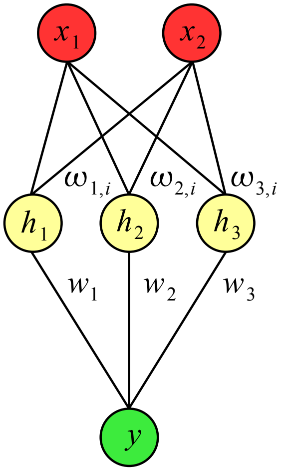
- Eingabewerte werden als Werte des Eingabe Layers benutzt
- Jedes Neuron im hidden Layer verarbeitet Eingabe
- Jedes Neuron im Ausgabe Layer verarbeitet Ausgaben des hidden Layer



Funktionsweise

Vorgehen um neuronales Netz anzuwenden:

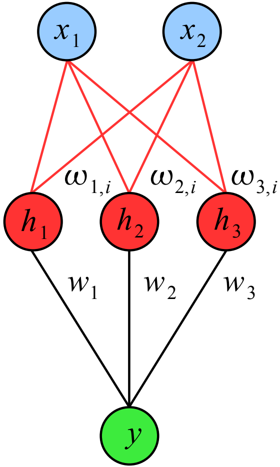
- Eingabewerte werden als Werte des Eingabe Layers benutzt
- Jedes Neuron im hidden Layer verarbeitet Eingabe
- Jedes Neuron im Ausgabe Layer verarbeitet Ausgaben des hidden Layer



Funktionsweise

Vorgehen um neuronales Netz anzuwenden:

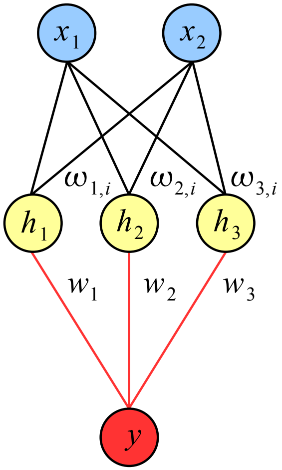
- Eingabewerte werden als Werte des Eingabe Layers benutzt
- Jedes Neuron im hidden Layer verarbeitet Eingabe
- Jedes Neuron im Ausgabe Layer verarbeitet Ausgaben des hidden Layer



Funktionsweise

Vorgehen um neuronales Netz anzuwenden:

- Eingabewerte werden als Werte des Eingabe Layers benutzt
- Jedes Neuron im hidden Layer verarbeitet Eingabe
- Jedes Neuron im Ausgabe Layer verarbeitet Ausgaben des hidden Layer



Und weiter?

Wir wissen jetzt, wie wir das Model anwenden können.

Aber

Wir wissen nicht, wie wir es trainieren können!

Klar:

Training des Modells über Anpassung der Gewichte. Aber wie gute Gewichtskombination finden?

Auf der Suche nach den optimalen Gewichten

- Gegeben initiale Gewichte w_i und $\omega_{i,j}$
- Desweiteren Trainingsdaten $< X, Y >$ mit Attributwerten X und Zielwerten Y
- Wiederhole für gesamte Daten mehrmals:
- Wähle zufälliges Trainingsbeispiel i
- Netz berechnet $f(x_i, W, \Omega)$ (Forward Schritt)
- Damit lässt sich dann Fehler bestimmen:

$$Err(x_i, y_i, W, \Omega) = (y_i - f(x_i, W, \Omega))^2$$

Naheliegende Idee: Partielles Differenzieren

- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k}$$

$$= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k}$$

$$= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k}$$

$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega)$$

$$= \delta_k \cdot h_k(x_i, \Omega)$$

Naheliegende Idee: Partielles Differenzieren

- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k}$$

$$= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k}$$

$$= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k}$$

$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega)$$

$$= \delta_k \cdot h_k(x_i, \Omega)$$

Naheliegende Idee: Partielles Differenzieren

- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

$$
\begin{aligned}
&\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k} \\
&= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k} \\
&= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k} \\
&= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega) \\
&= \delta_k \cdot h_k(x_i, \Omega)
\end{aligned}
$$

Naheliegende Idee: Partielles Differenzieren

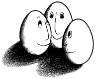
- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

$$
\begin{aligned}
&\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k} \\
&= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k} \\
&= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k} \\
&= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega) \\
&= \delta_k \cdot h_k(x_i, \Omega)
\end{aligned}
$$

Naheliegende Idee: Partielles Differenzieren

- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

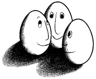
$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k}$$
$$= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k}$$
$$= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k}$$
$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega)$$
$$= \delta_k \cdot h_k(x_i, \Omega)$$



Naheliegende Idee: Partielles Differenzieren

- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k}$$
$$= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k}$$
$$= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k}$$
$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega)$$
$$= \delta_k \cdot h_k(x_i, \Omega)$$



Naheliegende Idee: Partielles Differenzieren

- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

$$
\begin{aligned}
&\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k} \\
&= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k} \\
&= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k} \\
&= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega) \\
&= \delta_k \cdot h_k(x_i, \Omega)
\end{aligned}
$$

Naheliegende Idee: Partielles Differenzieren

- Wir wollen über Wahl der Gewichte Fehler minimieren
- Zunächst beschränken auf Ausgabe Layer
- Ausgabewerte $h_j(x_i, \Omega)$ des hidden Layers aus Forward Schritt bekannt

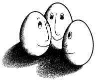
$$
\begin{aligned}
&\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial w_k} \\
&= \frac{\partial (y_i - f(x_i, W, \Omega))^2}{\partial w_k} \\
&= 2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \frac{-\partial \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))}{\partial w_k} \\
&= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_i, \Omega) \\
&= \delta_k \cdot h_k(x_i, \Omega)
\end{aligned}
$$

Und jetzt?

$$-2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot h_k(x_j, \Omega)$$

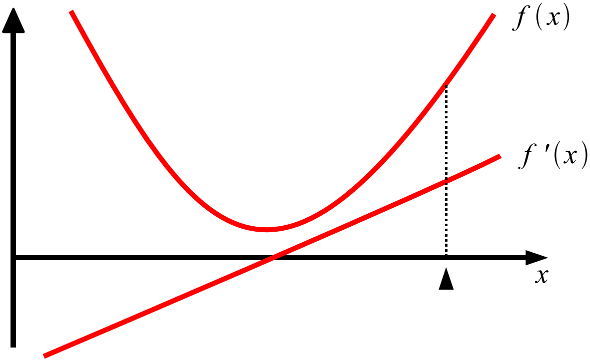
- Werte bereits aus Forward Schritt bekannt
- Können also leicht Steigung des Fehlers in Abhängigkeit von w_k an w_k berechnen
- (Hier differenzierbares σ nötig!)
- Wir können nun Gradientenabstieg mit Lernrate ν nutzen um Gewichte anzupassen:

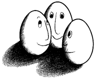
$$w'_k = w_k - \nu \cdot \delta_k \cdot h_j(x_j, \Omega)$$



Gradientenabstieg

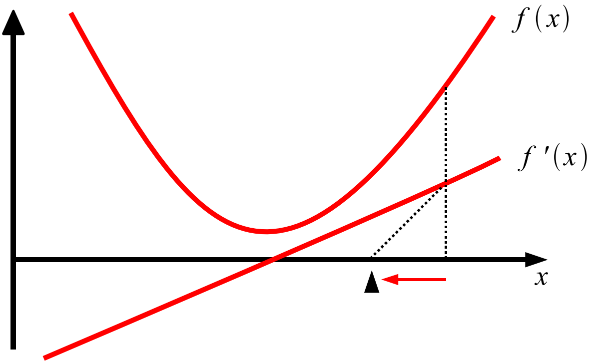
Schritte in Richtung des steilsten Abstiegs: Hier nur 1-dimensional

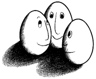




Gradientenabstieg

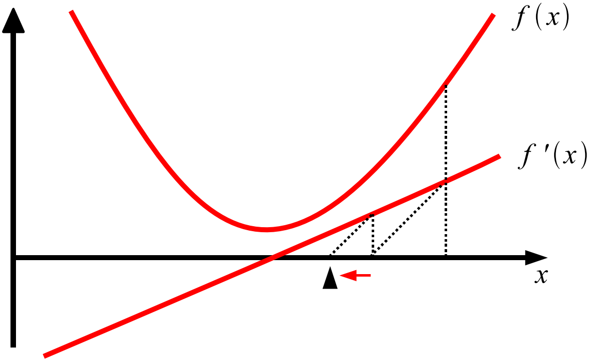
Schritte in Richtung des steilsten Abstiegs: Hier nur 1-dimensional

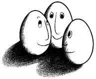




Gradientenabstieg

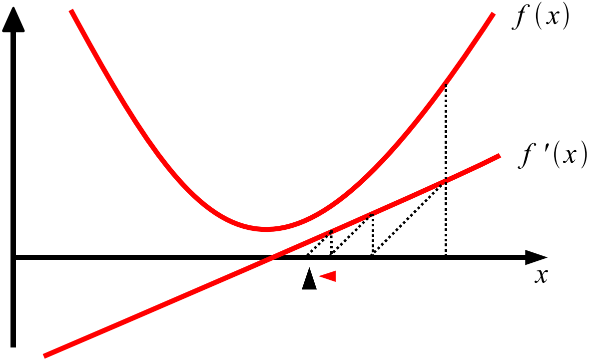
Schritte in Richtung des steilsten Abstiegs: Hier nur 1-dimensional





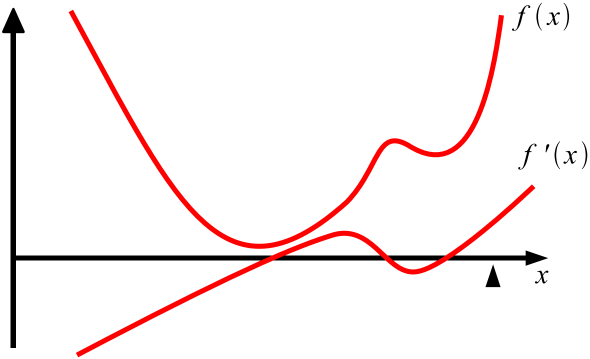
Gradientenabstieg

Schritte in Richtung des steilsten Abstiegs: Hier nur 1-dimensional



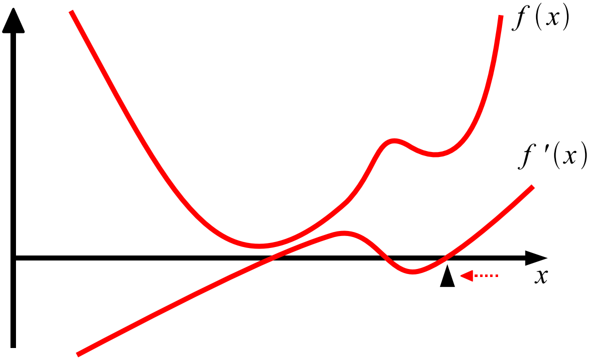
Problem

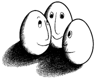
Gradientenabstieg ist Heuristik: Findet nicht garantiertes Optimum



Problem

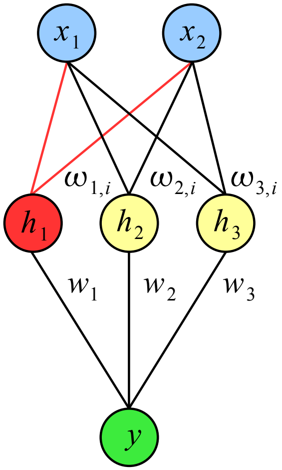
Gradientenabstieg ist Heuristik: Findet nicht garantiertes Optimum

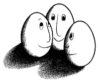




Wie hidden Layer behandeln?

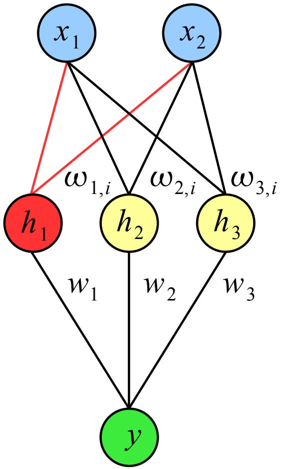
- Gewichte der Ausgabeschicht über Fehler optimiert
- Aber im hidden Layer kein Soll-Wert vorhanden
- Naiv: Wieder Fehler der Ausgabeschicht differenzieren, entsprechend nach ω

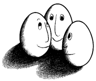




Wie hidden Layer behandeln?

- Gewichte der Ausgabeschicht über Fehler optimiert
- Aber im hidden Layer kein Soll-Wert vorhanden
- Naiv: Wieder Fehler der Ausgabeschicht differenzieren, entsprechend nach ω





Differenzieren, die zweite

$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial \omega_{w,m}}$$

$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

$$= \delta_k \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

- Im Prinzip mit w_k gewichteter Fehlergradient der Vorgängerschicht

Differenzieren, die zweite

$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial \omega_{w,m}}$$

$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

$$= \delta_k \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

- Im Prinzip mit w_k gewichteter Fehlergradient der Vorgängerschicht

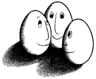
Differenzieren, die zweite

$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial \omega_{w,m}}$$

$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

$$= \delta_k \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

- Im Prinzip mit w_k gewichteter Fehlergradient der Vorgängerschicht



Differenzieren, die zweite

$$\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial \omega_{w,m}}$$

$$= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

$$= \delta_k \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

- Im Prinzip mit w_k gewichteter Fehlergradient der Vorgängerschicht

Differenzieren, die zweite

$$
\begin{aligned}
&\frac{\partial Err(x_i, y_i, W, \Omega)}{\partial \omega_{w,m}} \\
&= -2(y_i - \sigma(\sum_j w_j \cdot h_j(x_i, \Omega))) \cdot \sigma'(\sum_j w_j \cdot h_j(x_i, \Omega)) \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m} \\
&= \delta_k \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}
\end{aligned}
$$

- Im Prinzip mit w_k gewichteter Fehlergradient der Vorgängerschicht

Backpropagation Algorithmus

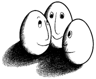
$$\omega'_{k,p} = \omega_{k,p} - \nu \cdot \delta_k \cdot w_k \cdot \sigma'(\sum_p \omega_{k,p} x_{i,p}) \cdot x_{i,m}$$

Dies liefert Idee für Algorithmus:

Der Backpropagation Algorithmus

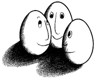
- Fehler in der Ausgabeschicht berechnen
- Gewichte der Ausgabeschicht anpassen
- Gradienten gewichten und als Fehler rückwärts propagieren

Outline



Idee

- Verwende Menge von unabhängigen Agenten, die zur Suchzeit Internet durchsuchen
- Agenten lernen dabei Relevanz von Links im Dokument zu erkennen
- Evolutionärer Ansatz soll Realisierung eines großen Suchraumes mit Beschränkung auf relevante Dokumente ermöglichen

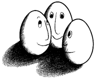


Probleme

- **Wie Relevanz von Dokument bestimmen?**
  Ohne Benutzerinteraktion schwer. Verwendung der Cosinus Ähnlichkeit aus der Theorie des Information Retrieval als Heuristik.
- **Wie Relevanz von Links bestimmen?**
  Links entspringen textuellem Kontext, entsprechend charakterisieren nahe Worte den Link besser und können entsprechend ihrer Nähe gewichtet werden.
- **Was heißt Nähe?**
  Begriff der Nähe unscharf und variabel mit der Textform. Verwende daher relatives Entfernungsmaß $dist(k, l)$ in Bezug auf einen Link l:
  Die Entfernung entspricht der Anzahl Links zwischen dem Keyword und dem Bezugslink.

Probleme

- **Wie Relevanz von Dokument bestimmen?**
  Ohne Benutzerinteraktion schwer. Verwendung der Cosinus Ähnlichkeit aus der Theorie des Information Retrieval als Heuristik.
- **Wie Relevanz von Links bestimmen?**
  Links entspringen textuellem Kontext, entsprechend charakterisieren nahe Worte den Link besser und können entsprechend ihrer Nähe gewichtet werden.
- **Was heißt Nähe?**
  Begriff der Nähe unscharf und variabel mit der Textform. Verwende daher relatives Entfernungsmaß $dist(k, l)$ in Bezug auf einen Link l:
  Die Entfernung entspricht der Anzahl Links zwischen dem Keyword und dem Bezugslink.



Probleme

- **Wie Relevanz von Dokument bestimmen?**
  Ohne Benutzerinteraktion schwer. Verwendung der Cosinus Ähnlichkeit aus der Theorie des Information Retrieval als Heuristik.
- **Wie Relevanz von Links bestimmen?**
  Links entspringen textuellem Kontext, entsprechend charakterisieren nahe Worte den Link besser und können entsprechend ihrer Nähe gewichtet werden.
- **Was heißt Nähe?**
  Begriff der Nähe unscharf und variabel mit der Textform. Verwende daher relatives Entfernungsmaß $dist(k, l)$ in Bezug auf einen Link l:
  Die Entfernung entspricht der Anzahl Links zwischen dem Keyword und dem Bezugslink.

Keyword-Link-Relevanz $rel(k,l)$

Die Relevanz eines Keywordvorkommen $k \in \mathscr{K}_i$ zu einem Link l entspricht:

$$rel(k,l) = \begin{cases} \frac{1}{dist(k,l)} & \text{falls } dist(k,l) \leq \rho \\ 0 & \text{sonst} \end{cases}$$

Distanz $dist(k,l)$

$dist(k,l) \sim$ Anzahl Links zwischen Keyword und Bezugslink inklusive

Beispiel

`xx key link xxx xxxxx xx link xxxx xx x xx xx bezug xx xxxx xxx`

$rel(key, bezug) = \frac{1}{3}$

Keyword-Link-Relevanz $rel(k, l)$

Die Relevanz eines Keywordvorkommen $k \in \mathscr{K}_i$ zu einem Link l entspricht:

$$rel(k,l) = \begin{cases} \frac{1}{dist(k,l)} & \text{falls } dist(k,l) \leq \rho \\ 0 & \text{sonst} \end{cases}$$

Distanz $dist(k, l)$

$dist(k, l) \sim$ Anzahl Links zwischen Keyword und Bezugslink inklusive

Beispiel

xx key link xxx xxxxx xx link xxxx xx x xx xx bezug xx xxxx xxx

$rel(key, bezug) = \frac{1}{3}$

Keyword-Link-Relevanz $rel(k, l)$

Die Relevanz eines Keywordvorkommen $k \in \mathscr{K}_i$ zu einem Link l entspricht:

$$rel(k,l) = \begin{cases} \frac{1}{dist(k,l)} & \text{falls } dist(k,l) \leq \rho \\ 0 & \text{sonst} \end{cases}$$

Distanz $dist(k, l)$

$dist(k,l) \sim$ Anzahl Links zwischen Keyword und Bezugslink inklusive

Beispiel

`xx key link xxx xxxxx xx link xxxx xx x xx xx bezug xx xxxx xxx`

$rel(key, bezug) = \frac{1}{3}$

Outline

Menczers Agenten

Agenten sind autonome Instanzen eines Programms, die jeweils über diese Daten verfügen:

- **Energiezähler**
  Gibt die Energie an, die dem Agenten zur Verfügung steht. Energie wird verbraucht zum Durchführen verschiedener Aktionen.
- **neuronales Netz**
  3-schichtiges neuronales Netz zur Regression. Hiermit erfolgt Schätzung der Relevanz von Links anhand von Keyword-Relevanz-Vektoren.
- **Keywordvektor**
  Vektor der Suchworte

Outline

1. Motivation
   - Eine kleine Geschichte des Information Retrieval
2. Ansätze
   - Information Retrieval
   - Indexsuchmaschinen
   - Neuronale Netze
3. Menczers Agents
   - Idee
   - Die Agenten
   - Der Algorithmus
4. Ergebnisse
   - Probleme
   - Neue Maße
   - Ergebnisse

Der Algorithmus - 1

1. Initialisiere:
   1. Kreiere eine Menge $\mathscr{A}$ von Agenten auf Startseiten. Als Startseiten lassen sich nutzen:
      - statische Bookmarks
      - Suchergebnisse von Indexsuchmaschinen
   2. Initialisiere neuronale Netze mit zufälligen Gewichten
   3. Als Keywordvektoren benutze TF-Vektor der Anfrage

Der Algorithmus - 2

2. Führe dann parallel für alle Agenten A_i durch:
   1. Extrahiere aus Seite Positionen $\mathscr{K}_i$ aller Keywords in K_i und Links L
   2. Berechne für jeden Link $l \in L$ Keyword-Relevanz-Vektor d_l:

$$d_l = \begin{pmatrix} \sum_{k \in \mathscr{K}_1} rel(k,l) \\ \vdots \\ \sum_{k \in \mathscr{K}_{|K|}} rel(k,l) \end{pmatrix}$$

Der Algorithmus - 2

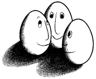
2. Führe dann parallel für alle Agenten A_i durch:
   1. Extrahiere aus Seite Positionen $\mathscr{K}_i$ aller Keywords in K_i und Links L
   2. Berechne für jeden Link $l \in L$ Keyword-Relevanz-Vektor d_l:

$$d_l = \begin{pmatrix} \sum_{k \in \mathscr{K}_1} rel(k, l) \\ \vdots \\ \sum_{k \in \mathscr{K}_{|K|}} rel(k, l) \end{pmatrix}$$

Beispiel

`xx key link xxx xxxxx xx link xxxx xx x key xx bezug xx xxxx key`

$\sum_{key_i \in \mathscr{K}_m} rel(key_i, bezug) = ?$



Der Algorithmus - 2

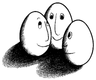
2. Führe dann parallel für alle Agenten A_i durch:
   1. Extrahiere aus Seite Positionen $\mathscr{K}_i$ aller Keywords in K_i und Links L
   2. Berechne für jeden Link $l \in L$ Keyword-Relevanz-Vektor d_l:

$$d_l = \begin{pmatrix} \sum_{k \in \mathscr{K}_1} rel(k, l) \\ \vdots \\ \sum_{k \in \mathscr{K}_{|K|}} rel(k, l) \end{pmatrix}$$

Beispiel

`xx key link xxx xxxxx xx link xxxx xx x key xx bezug xx xxxx key`

$\sum_{key_i \in \mathscr{K}_m} rel(key_i, bezug) = \frac{1}{3}$



Der Algorithmus - 2

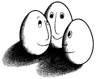
2. Führe dann parallel für alle Agenten A_i durch:
    1. Extrahiere aus Seite Positionen $\mathscr{K}_i$ aller Keywords in K_i und Links L
    2. Berechne für jeden Link $l \in L$ Keyword-Relevanz-Vektor d_l:

$$d_l = \begin{pmatrix} \sum_{k \in \mathscr{K}_1} rel(k, l) \\ \vdots \\ \sum_{k \in \mathscr{K}_{|K|}} rel(k, l) \end{pmatrix}$$

Beispiel

`xx key link xxx xxxxx xx link xxxx xx x key xx bezug xx xxxx key`

$\sum_{key_i \in \mathscr{K}_m} rel(key_i, bezug) = \frac{1}{3} + \frac{1}{1}$



Der Algorithmus - 2

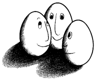
2. Führe dann parallel für alle Agenten A_i durch:
   1. Extrahiere aus Seite Positionen $\mathscr{K}_i$ aller Keywords in K_i und Links L
   2. Berechne für jeden Link $l \in L$ Keyword-Relevanz-Vektor d_l:

$$d_l = \begin{pmatrix} \sum_{k \in \mathscr{K}_1} rel(k, l) \\ \vdots \\ \sum_{k \in \mathscr{K}_{|K|}} rel(k, l) \end{pmatrix}$$

Beispiel

`xx key link xxx xxxxx xx link xxxx xx x key xx bezug xx xxxx key`

$\sum_{key_i \in \mathscr{K}_m} rel(key_i, bezug) = \frac{1}{3} + \frac{1}{1} + \frac{1}{1}$



Der Algorithmus - 3

2. Fortsetzung:
    3. Benutze d_l als Eingabe für neuronales Netz, dieses schätzt Relevanz λ_l des Links l
    4. Wähle zufällig einen Link nach Verteilung:

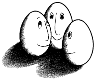
$$Pr(l) = \frac{e^{\beta \lambda_l}}{\sum_{l' \in \mathscr{L}} e^{\beta \lambda_{l'}}}$$

Der Algorithmus - 4

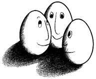
2. Fortsetzung:
   5. Lade Seite S_l des gewählten Links l mit Energiekosten c
   6. Berechne mit Cosinus Ähnlichkeit zwischen TF-Vektoren der Seite S_l und der Suchanfrage “tatsächliche” Relevanz $Rel(S_l)$ der Seite.
   7. Erhöhe Energie um $Rel(S_l)$
   8. Trainiere neuronales Netz nach:
      Verwende $Rel(S_l)$ als Ziel mit alter Eingabe d_l

Der Algorithmus - 5

3. Sterben oder Vermehrung der Agenten
    1. Lösche alle Agenten ohne Energie
    2. Verfügt ein Agent A_i über genug Energie, kann er θ bezahlen um sich fortzupflanzen:
        - Erzeuge neuen Agenten $A_{i'}$ auf der aktuellen Seite
        - Addiere Rauschen auf zufällige Teilmenge der Gewichte seines neuronalen Netzes
        - Füge häufigstes Keyword der Seite $S_l(A_i)$ zum Keywordvektor von Agent $A_{i'}$ hinzu
4. Wiederhole Schritte 2 und 3 solange, bis
    - kein Agent mehr lebt
    - der Benutzer abbricht



Outline



Performancebeurteilung

Um Suchergebnisse zu bewerten, müssen verschiedene Aspekte betrachtet werden:

- Wie viele relevante Dokumente wurden gefunden?
- Wie hoch ist der Anteil an relevanten Dokumenten?
- Wie ist die Reihenfolge im Suchergebnis?
- Wie aktuell sind die gefundenen Dokumente?

Problem:

Eingeführte Maße Recall und Precision nicht nutzbar:

- Gesamtmenge nicht bekannt
- Relevante Teilmenge nicht bekannt
- Relevanz nur durch Benutzer bestimmbar
- Aktualität wird nicht betrachtet

Performancebeurteilung

Um Suchergebnisse zu bewerten, müssen verschiedene Aspekte betrachtet werden:

- Wie viele relevante Dokumente wurden gefunden?
- Wie hoch ist der Anteil an relevanten Dokumenten?
- Wie ist die Reihenfolge im Suchergebnis?
- Wie aktuell sind die gefundenen Dokumente?

Problem:

Eingeführte Maße Recall und Precision nicht nutzbar:

- Gesamtmenge nicht bekannt
- Relevante Teilmenge nicht bekannt
- Relevanz nur durch Benutzer bestimmbar
- Aktualität wird nicht betrachtet

Outline

Neue Performancemaße

Um diese Probleme zu umgehen, werden neue Maße definiert, die Relevanz anhand der Cosinus Ähnlichkeit messen:

- Estimated Precision
- Estimated Recall
- Estimated Recency

Estimated Precision

Sei eine Suchanfrage q gegeben und dazu eine Ergebnismenge C_q sowie ein Vektor r_q der relevanten Worte. Dann ist die *estimated precision*:

$$P(C_q) \quad \equiv \quad \frac{1}{|C_q|} \dot{\sum}_{p \in C_q} sim(r_q, p)$$

Zur Erinnerung: Precision

$$\text{Precision} = \frac{|\{\text{relevante Dokumente}\} \cap \{\text{gefundene Dokumente}\}|}{|\{\text{gefundene Dokumente}\}|}$$

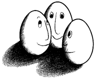
Estimated Recall

Sei eine Suchanfrage q gegeben und dazu eine Ergebnismenge C_q sowie ein Vektor r_q der relevanten Worte. Dann ist der *estimated recall*:

$$R(C_q) \equiv \sum_{p \in C_q} sim(r_q, p)$$

Zur Erinnerung: Recall

$$\text{Recall} = \frac{|\{\text{relevante Dokumente}\} \cap \{\text{gefundene Dokumente}\}|}{|\{\text{relevante Dokumente}\}|}$$



Estimated Recency

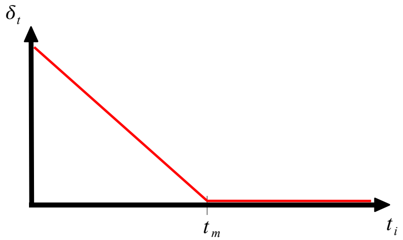
Sei eine Suchanfrage q gegeben und dazu eine Ergebnismenge C_q. Ordne $t_m(p)$ und $t_i(p)$ jeder Seite p das Datum der letzten Modifizierung bzw. der letzten Indizierung zu. Dann ist die *estimated recency*:

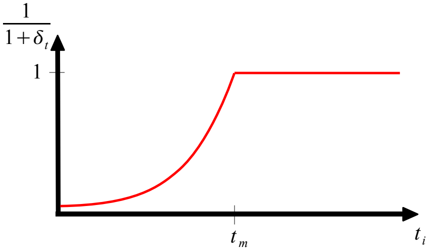
$$T(C_q) \quad \equiv \quad \frac{1}{|C_q|} \dot{\sum}_{p \in C_q} \frac{1}{1 + \delta_t(p)}$$

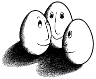
mit

$$\delta_t(p) = \begin{cases} t_m(p) - t_i(p) & \text{falls } t_m(p) > t_i(p) \\ 0 & \text{sonst} \end{cases}$$

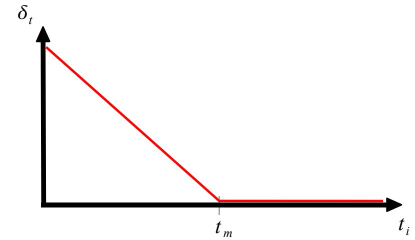
Beispiel:

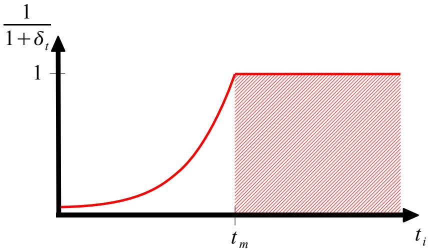




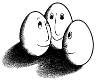


Beispiel:





Outline



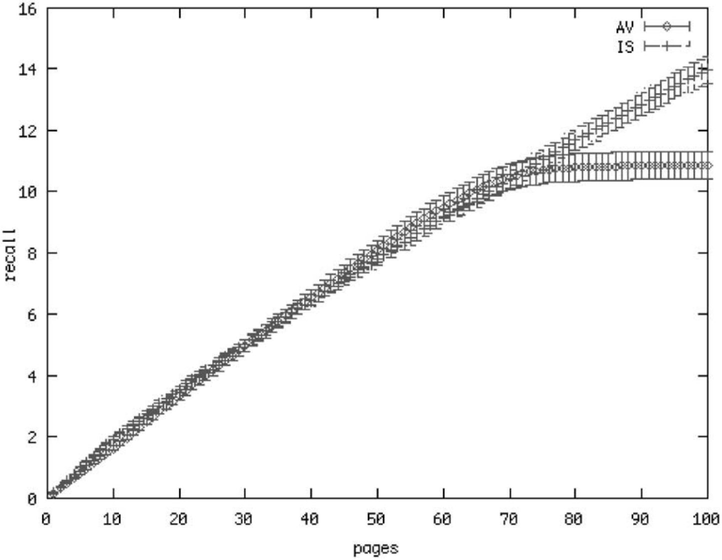
Die Testumgebung

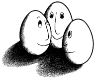
- Vergleich MySpiders mit AltaVista
- Benutze 100 Yahoo Leaf-Topics mit je mehr als 10 externen Links als Suchanfrage
- Initialisiere MySpiders mit 100 top ranked AltaVista Suchergebnissen
- Entferne alle Yahoo Seiten aus den Ergebnissen
- Benutze Yahoos Textbeschreibung als r_q

Vergleiche dann mittels der Heuristiken:

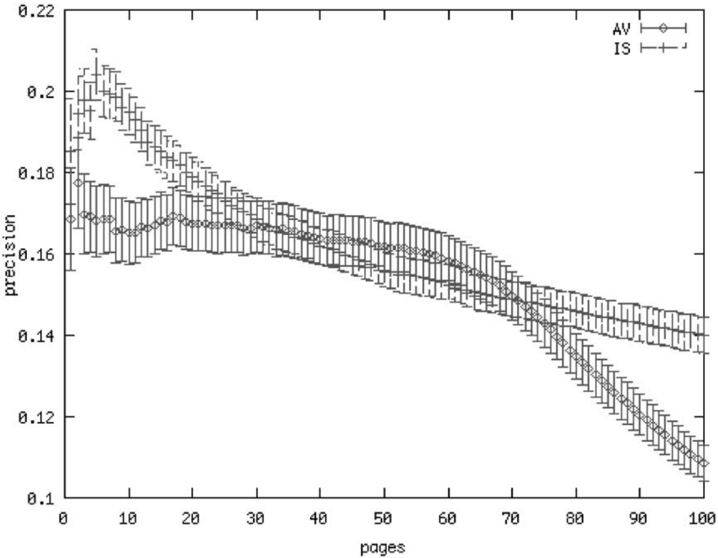
- Estimated Precision
- Estimated Recall
- Estimated Recency

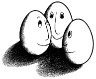
Estimated Recall $\sim \frac{\textit{gefundene Relevante}}{\textit{Relevante}}$



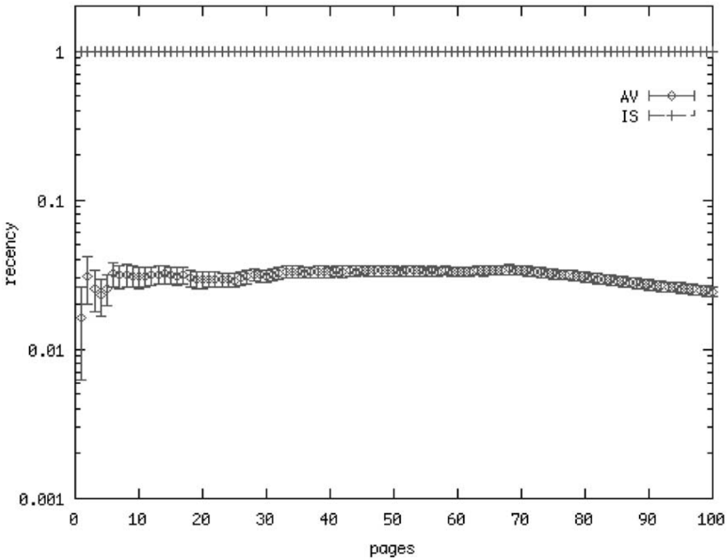


Estimated Precision $\sim \frac{\textit{gefundene Relevante}}{\textit{Gefundene}}$





Estimated Recency



Literaturhinweise

Filippo Menczer
Complementing search engines with online web mining agtens
Decision Support Systems, 35(2003):195–212

Tom M. Mitchell
Machine Learning
McGraw-Hill, 1997

Trevor Hastie, et al
The Elements of Statistical Learning
Springer, 2001