"30 "31 "32 "33 "3B "3C "3D "01 "02 "10 "03 "04 "11 "3E "3F
"1D "18 "23 "16 "72 "73 "74 "75 "76

Experimente, Evaluierung und Tools

Helmut Schmid

Centrum für Informations- und Sprachverarbeitung
Ludwig-Maximilians-Universität München

Stand: 22. Januar 2025

Organisatorisches

- Dienstag, 10-12 Uhr c.t.

  Vorlesung:

  - Vermittlung/Wiederholung der theoretischen Grundlagen
  - Besprechung der Details der Aufgaben

- Mittwoch, 14-16 Uhr c.t.

  Übungen: praktische Aufgaben zur

  - Anwendung vorhandener Werkzeuge
  - Entwicklung eigener Programme
  - Ort: Rechnerpool Gobi

- statt schriftlicher Prüfung: Benotung der abgegebenen Aufgaben
- Die Aufgaben dürfen maximal zu dritt bearbeitet werden.
- Alle wichtigen Informationen zu dem Kurs sind auf der Kursseite verfügbar, die über meine CIS-Homepage erreichbar ist.

Überblick

- Korpora

  Crawling, Vorverarbeitung, Tokenisierung
- Morphologie

  Finite-State-Transducer, Flexionsklassen, morphophonologische Regeln
- Spam-Erkennung

  Naive Bayes, log-lineare Modelle
- Wortart-Tagging

  Conditional Random Fields
- Spam-Erkennung

  Neuronale Netze, PyTorch
- Parsing

  Neuronale Netze

Korpora

bilden die Grundlage der Forschung in der Computerlinguistik

Korpusquellen:

- Bücher (z.B. Gutenberg-Archiv)
- Zeitungen (Zeit, FAZ, TAZ)
- Wikipedia
  - großer Umfang ($>$ 1,74 Milliarden Wörter)
  - 264 Sprachen
- soziale Medien (Twitter)
  - sehr großer Umfang
  - sehr aktuell
  - Tippfehler, Grammatikfehler, Slang, Abkürzungen
- Parallelkorpora (EU, UN, kanadisches Parlament, Handbücher)

Vor ihrer Nutzung müssen die Korpora erst aufbereitet werden.

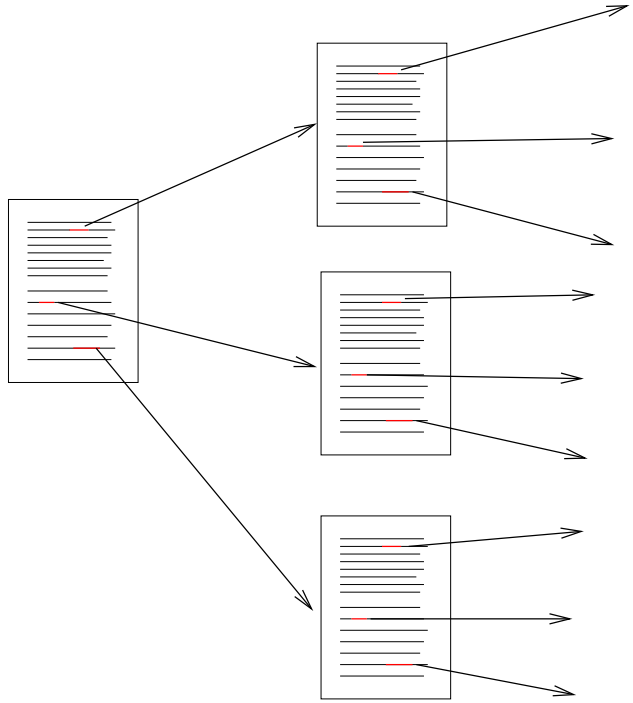
Korpusaufbereitung

Schritte

- Konvertierung von PDF-, DOC-, HTML-Dateien etc. in reine Textdateien
- Entfernung von nicht-relevanten Teilen (Bilder, Tabellen etc.)
- falls nötig Konvertierung in Unicode (UTF8)
- Tokenisierung
  Zerlegung in Sätze und Tokens
- linguistische Annotation
  mit Wortarten, Lemmata, Parsebäumen, Eigennamen etc.

Textsammlung durch Web-Crawling

- Ein Crawler durchwandert das Internet systematisch.
- Suchmaschinen erstellen mit Crawling ihren Suchindex.
- Crawling kann auch zum Aufbau von Webkorpora genutzt werden.



- Prinzip
  - Herunterladen einer Startseite
  - Extraktion der Verweise auf weitere Seiten
  - Herunterladen der verlinkten Seiten
  - Extraktion der Verweise usw.

Crawling-Werkzeuge

- BootCat: komplexes Werkzeug zur Erstellung von Webkorpora, verwendet Suchmaschinenanfragen, um themenspezifische Seiten herunterzuladen
- wget: Programm zum rekursiven Herunterladen von Webseiten
  Beispiel: wget -r -w 1 www.bbc.com

  $\rightarrow$ benutzt in Aufgabe 1

PDF, DOC und HTML

TV-Duell

Trump geht die Puste aus

90 Minuten gegen Hillary Clinton sind zu viel für Donald Trump. Nicht nur ahnungslos, auch unkonzentriert blamiert er sich vor Millionenpublikum. Ist die Wahl gelaufen?

Ein Kommentar von **Paul Middelhoff**, Washington D.C.

$\Rightarrow$

TV-Duell
Trump geht die Puste aus
90 Minuten gegen Hillary Clinton sind zu viel für Donald Trump. Nicht nur ahnungslos, auch unkonzentriert blamiert er sich vor Millionenpublikum. Ist die Wahl gelaufen?
Ein Kommentar von Paul Middelhoff, Washington D.C.

- Texte liegen oft formatiert vor.
- Für Korpora werden die reinen Texte benötigt.
- Formatierte Texte müssen daher umgewandelt werden:
  - PDF-Dateien: pdftotext (Ergebnis nicht immer gut)
  - DOC-Dateien: mit Word, LibreOffice etc.
  - HTML-Datei: oft spezifische Lösungen notwendig, um irrelevante Teile der Seite (Werbung, Links etc.) auszufiltern ($\rightarrow$ Aufgabe 1)
  - Spezialfall Bild: OCR

Textextraktion aus HTML-Seiten

...

<h1 class="article-heading" itemprop="headline">
Künstliche Intelligenz<span
class="visually-hidden">: <span
class="article-heading__title">Watson, wir haben ein
Problem</h1>

Lesen, schreiben, zuhören und verstehen – intelligente Maschinen können immer
mehr Dinge, die bisher nur Menschen konnten. Was bedeutet das für unsere
Jobs? Und für uns?
Von<span itemprop="author" itemscope
itemtype="http://schema.org/Person">
<a href="http://www.zeit.de/autoren/G/Lars_Gaede" itemprop="url"
data-vars-url="www.zeit.de/autoren/G/Lars_Gaede">
Lars Gaede

...

Textextraktion aus HTML-Seiten

...

<h1 class="article-heading" itemprop="headline">
Künstliche Intelligenz<span
class="visually-hidden">: <span
class="article-heading__title">Watson, wir haben ein
Problem</h1>

Lesen, schreiben, zuhören und verstehen – intelligente Maschinen können immer
mehr Dinge, die bisher nur Menschen konnten. Was bedeutet das für unsere
Jobs? Und für uns?
Von<span itemprop="author" itemscope
itemtype="http://schema.org/Person">
<a href="http://www.zeit.de/autoren/G/Lars_Gaede" itemprop="url"
data-vars-url="www.zeit.de/autoren/G/Lars_Gaede">
Lars Gaede

...

Zeichensatzkonvertierung

- Texte können unterschiedlich kodiert sein: ISO-8859-1 (Latin1), Windows-1252 etc.
- Ein Korpus sollte einen einheitlichen Zeichensatz verwenden
- Unicode erlaubt die Darstellung (fast) aller Zeichen
  $\Rightarrow$ Umwandlung aller Texte nach Unicode (meist UTF8)
- Linux-Werkzeuge für Zeichensatzkonvertierung
  - recode
  - iconv (ähnlich, aber andere Aufruf-Syntax)

Zeichensatzkonvertierung

Anwendung von recode

```
sh-4.2$ cat test.txt
sch□ne Gr□□e,
  G□nther
sh-4.2$ file -bi test.txt
text/plain; charset=iso-8859-1
sh-4.2$ recode latin1..utf8 test.txt
sh-4.2$ cat test.txt
schöne Grüße,
  Günther
```

Tokenisierung

- Für die Computerlinguistik sind der Satz und das Wort wichtige Einheiten, da viele CL-Programme auf Sätzen und Wörtern operieren.
- Den ersten Schritt der linguistischen Annotation bildet daher die Tokenisierung = Zerlegung in Sätze und Wörter

Tokenisierung – Schritte

„Sie fliegt nach London, New York usw.", sagte er.

- Bei Leerzeichen Wortgrenzen einfügen
- Satzzeichen (!?;:), Anführungszeichen (,,"), Klammern, Klitika (hat's) als separate Tokens abtrennen
- Abkürzungen erkennen und als Einheit behandeln
- Mehrwortausdrücke erkennen
- Satzgrenzen markieren

→ Aufgabe 1

Tokenisierung – Schritte

„Sie fliegt nach London, New York usw.", sagte er.

- Bei Leerzeichen Wortgrenzen einfügen

  „Sie | fliegt | nach | London, | New | York | usw.", | sagte | er.

- Satzzeichen (!?.;:,), Anführungszeichen („"), Klammern, Klitika (hat's) als separate Tokens abtrennen

  „ | Sie | fliegt | nach | London | , | New | York | usw | . | " | , | sagte | er | .

- Abkürzungen erkennen und als Einheit behandeln

  „ | Sie | fliegt | nach | London | , | New | York | usw. | " | , | sagte | er | .

- Mehrwortausdrücke erkennen

  „ | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | .

- Satzgrenzen markieren

  ⟨s⟩ „ | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | . ⟨/s⟩

→ Aufgabe 1

Tokenisierung – Schritte

,,Sie fliegt nach London, New York usw.", sagte er.

- Bei Leerzeichen Wortgrenzen einfügen

  ,,Sie | fliegt | nach | London, | New | York | usw.", | sagte | er.

- Satzzeichen (!?.;:,), Anführungszeichen (,,'), Klammern, Klitika (hat's) als separate Tokens abtrennen

  ,, | Sie | fliegt | nach | London | , | New | York | usw | . | " | , | sagte | er | .

- Abkürzungen erkennen und als Einheit behandeln

  ,, | Sie | fliegt | nach | London | , | New | York | usw. | " | , | sagte | er | .

- Mehrwortausdrücke erkennen

  ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | .

- Satzgrenzen markieren

  ⟨*s*⟩ ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | . ⟨/*s*⟩

→ Aufgabe 1

Tokenisierung – Schritte

,,Sie fliegt nach London, New York usw.", sagte er.

- Bei Leerzeichen Wortgrenzen einfügen

  ,,Sie | fliegt | nach | London, | New | York | usw.", | sagte | er.

- Satzzeichen (!?.;:,), Anführungszeichen (,,'), Klammern, Klitika (hat's) als separate Tokens abtrennen

  ,, | Sie | fliegt | nach | London | , | New | York | usw | . | " | , | sagte | er | .

- Abkürzungen erkennen und als Einheit behandeln

  ,, | Sie | fliegt | nach | London | , | New | York | usw. | " | , | sagte | er | .

- Mehrwortausdrücke erkennen

  ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | .

- Satzgrenzen markieren

  ⟨*s*⟩ ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | . ⟨/*s*⟩

→ Aufgabe 1

Tokenisierung – Schritte

,,Sie fliegt nach London, New York usw.", sagte er.

- Bei Leerzeichen Wortgrenzen einfügen

  ,,Sie | fliegt | nach | London, | New | York | usw.", | sagte | er.

- Satzzeichen (!?.;:,), Anführungszeichen (,,'), Klammern, Klitika (hat's) als separate Tokens abtrennen

  ,, | Sie | fliegt | nach | London | , | New | York | usw | . | " | , | sagte | er | .

- Abkürzungen erkennen und als Einheit behandeln

  ,, | Sie | fliegt | nach | London | , | New | York | usw. | " | , | sagte | er | .

- Mehrwortausdrücke erkennen

  ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | .

- Satzgrenzen markieren

  ⟨s⟩ ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | . ⟨/s⟩

→ Aufgabe 1

Tokenisierung – Schritte

,,Sie fliegt nach London, New York usw.", sagte er.

- Bei Leerzeichen Wortgrenzen einfügen

  ,,Sie | fliegt | nach | London, | New | York | usw.", | sagte | er.
- Satzzeichen (!?.;:,), Anführungszeichen (,,'), Klammern, Klitika (hat's) als separate Tokens abtrennen

  ,, | Sie | fliegt | nach | London | , | New | York | usw | . | " | , | sagte | er | .
- Abkürzungen erkennen und als Einheit behandeln

  ,, | Sie | fliegt | nach | London | , | New | York | usw. | " | , | sagte | er | .
- Mehrwortausdrücke erkennen

  ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | .
- Satzgrenzen markieren

  ⟨*s*⟩ ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | . ⟨/*s*⟩

→ Aufgabe 1

Tokenisierung – Schritte

,,Sie fliegt nach London, New York usw.", sagte er.

- Bei Leerzeichen Wortgrenzen einfügen

  ,,Sie | fliegt | nach | London, | New | York | usw.", | sagte | er.
- Satzzeichen (!?.;:,), Anführungszeichen (,,'), Klammern, Klitika (hat's) als separate Tokens abtrennen

  ,, | Sie | fliegt | nach | London | , | New | York | usw | . | " | , | sagte | er | .
- Abkürzungen erkennen und als Einheit behandeln

  ,, | Sie | fliegt | nach | London | , | New | York | usw. | " | , | sagte | er | .
- Mehrwortausdrücke erkennen

  ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | .
- Satzgrenzen markieren

  ⟨*s*⟩ ,, | Sie | fliegt | nach | London | , | New York | usw. | " | , | sagte | er | . ⟨/*s*⟩

→ Aufgabe 1

Wort-Segmentierung

Viele Sprachen markieren keine Wortgrenzen.

Chinesisch: 交通运输部正在研究调整重大节日高速路免费通行政策

Japanisch: 筋肉成分が足りないと感じる方へ。ロコモア。今なら1000円+税でお試し

Thai: ไปตรวจสอบข้าวของกระจุยกระจายเกลื่อนพื้นเป็นจำนวนมาก รวมทั้งของที่จะส่งให้ลูกค้า

Koreanisch: 스헬기의 동체 바로 근처"라며 "나머지 실종자들에 대

Wort-Segmentierung

Deutscher Satz ohne Leerzeichen: erbestellteinbierimgasthaus

Die bisherigen Tokenisierungs-Heuristiken funktionieren nicht, stattdessen

- longest Match mit großen Wortlisten
  erbe | stellte | in | bier | im | gasthaus
  Was ist hier schiefgegangen?
- N-Gramm-Sprachmodelle zur Desambiguierung
  er | bestellt | ein | bier | im | gasthaus
- Probleme mit unbekannten Wörtern
  er | bestellt | ein | bier | im | n | epo | m | u | k
- Tagging-Ansatz
  e r b e s t e l l t e i n b i e r i m g a s t h a u s
  1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0
- Es gibt keine allgemeingültige Definition von "Wort"
  (vgl. computer screen vs. Computerbildschirm)

Wort-Segmentierung

Deutscher Satz ohne Leerzeichen: erbestellteinbierimgasthaus

Die bisherigen Tokenisierungs-Heuristiken funktionieren nicht, stattdessen

- longest Match mit großen Wortlisten
  erbe | stellte | in | bier | im | gasthaus
  Was ist hier schiefgegangen?
- N-Gramm-Sprachmodelle zur Desambiguierung
  er | bestellt | ein | bier | im | gasthaus
- Probleme mit unbekannten Wörtern
  er | bestellt | ein | bier | im | n | epo | m | u | k
- Tagging-Ansatz
  e r b e s t e l l t e i n b i e r i m g a s t h a u s
  1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0
- Es gibt keine allgemeingültige Definition von "Wort"
  (vgl. computer screen vs. Computerbildschirm)

Wort-Segmentierung

Deutscher Satz ohne Leerzeichen: erbestellteinbierimgasthaus

Die bisherigen Tokenisierungs-Heuristiken funktionieren nicht, stattdessen

- longest Match mit großen Wortlisten
  erbe | stellte | in | bier | im | gasthaus
  Was ist hier schiefgegangen?
- N-Gramm-Sprachmodelle zur Desambiguierung
  er | bestellt | ein | bier | im | gasthaus
- Probleme mit unbekannten Wörtern
  er | bestellt | ein | bier | im | n | epo | m | u | k
- Tagging-Ansatz
  e r b e s t e l l t e i n b i e r i m g a s t h a u s
  1 0 1 0 0 0 0 0 0 0 1 0 0 1 0 0 0 1 0 1 0 0 0 0 0 0
- Es gibt keine allgemeingültige Definition von "Wort"
  (vgl. computer screen vs. Computerbildschirm)

Wort-Segmentierung

Deutscher Satz ohne Leerzeichen: erbestellteinbierimgasthaus

Die bisherigen Tokenisierungs-Heuristiken funktionieren nicht, stattdessen

- longest Match mit großen Wortlisten
  erbe | stellte | in | bier | im | gasthaus
  Was ist hier schiefgegangen?
- N-Gramm-Sprachmodelle zur Desambiguierung
  er | bestellt | ein | bier | im | gasthaus
- Probleme mit unbekannten Wörtern
  er | bestellt | ein | bier | im | n | epo | m | u | k
- Tagging-Ansatz

e	r	b	e	s	t	e	l	l	t	e	i	n	b	i	e	r	i	m	g	a	s	t	h	a	u	s
1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0

- Es gibt keine allgemeingültige Definition von "Wort"
  (vgl. computer screen vs. Computerbildschirm)

Wort-Segmentierung

Deutscher Satz ohne Leerzeichen: erbestellteinbierimgasthaus

Die bisherigen Tokenisierungs-Heuristiken funktionieren nicht, stattdessen

- longest Match mit großen Wortlisten
  erbe | stellte | in | bier | im | gasthaus
  Was ist hier schiefgegangen?
- N-Gramm-Sprachmodelle zur Desambiguierung
  er | bestellt | ein | bier | im | gasthaus
- Probleme mit unbekannten Wörtern
  er | bestellt | ein | bier | im | n | epo | m | u | k
- Tagging-Ansatz

e	r	b	e	s	t	e	l	l	t	e	i	n	b	i	e	r	i	m	g	a	s	t	h	a	u	s
1	0	1	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	0	0

- Es gibt keine allgemeingültige Definition von “Wort”
  (vgl. computer screen vs. Computerbildschirm)

Linguistische Annotation

Für die linguistische Forschung und die Sprachverarbeitung werden Korpora linguistisch annotiert

- Tokenisierung
- morphologische Analyse der Wörter ($\rightarrow$ nächstes Thema)
- Wortart-Annotation
- Lemmatisierung
- syntaktische Annotation (Parsing)
- Erkennung von Namen
- ...

Morphologie

Morphologie untersucht die Struktur von Wörtern

- Zerlegung in Morpheme: Formel-ab-leit-ung-en
  Morpheme sind die kleinsten bedeutungstragenden Einheiten

- weitergehende Analyse:

Basislemma	leiten/V
Präfigierung	ab/PART leiten/V
Nominalisierung	ab/PART leiten/V ung/NN
Komposition	Formel/NN ab/PART leiten/V ung/NN
Flexion	Formel/NN ab/PART leiten/V ung/NN en/PI

Morphologische Prozesse

Flexion

- Flexionsmorpheme markieren syntaktische Eigenschaften
  rechn-est 2. Sg. Präsens
  ge-rechn-et Partizip Perfekt
  rechn-en Infinitiv, 1. Pl. Präsens, 3. Pl. Präsens
- Die Flexionsmorpheme werden an den Wortstamm angefügt
- Flexion von Verben: Konjugation
- Flexion von Nomen, Adjektiven: Deklination

Deutsch ist eine fusionale Sprache, d.h. ein Morphem repräsentiert mehrere syntaktische Merkmale → "-est" - 2. Sg. Präsens

Deutsch zeigt außerdem Synkretismus, d.h. eine Form hat mehrere Analysen → rechnen

Morphologische Prozesse

Flexion

- Flexionsmorpheme markieren syntaktische Eigenschaften
  rechn-est 2. Sg. Präsens
  ge-rechn-et Partizip Perfekt
  rechn-en Infinitiv, 1. Pl. Präsens, 3. Pl. Präsens
- Die Flexionsmorpheme werden an den Wortstamm angefügt
- Flexion von Verben: Konjugation
- Flexion von Nomen, Adjektiven: Deklination

Deutsch ist eine fusionale Sprache, d.h. ein Morphem repräsentiert mehrere syntaktische Merkmale $\rightarrow$ "-est" - 2. Sg. Präsens

Deutsch zeigt außerdem Synkretismus, d.h. eine Form hat mehrere Analysen $\rightarrow$ rechnen

Morphologische Prozesse

Türkisch hat eine sehr komplexe Flexion (agglutinierende Sprache)

pişirdiler - they caused it to be cooked

piş	to cook (Stamm)
ir	Kausativ
di	Vergangenheit
ler	3. Person Plural

Morphologische Prozesse

Derivation

Ableitung neuer Wortformen mit anderer Wortart oder Bedeutung aus vorhandenen Wortformen

setzen → übersetzen → übersetzbar → unübersetzbar → Unübersetzbarkeit

- über- Verbpräfix
- -bar leitet ein Adjektiv aus einem Verb ab
- un- negierendes Adjektivpräfix
- keit leitet ein Nomen aus einem Adjektiv ab

Derivationsmorpheme können auch leer sein: wohnen → das Wohnen (Konversion)

Neoklassische Wortbildung

- existiert parallel zur normalen deutschen Wortbildung
- in vielen europäischen Sprachen
- Grund: Latein war lange die Sprache der Wissenschaft. Lateinische (und griechische) Fachbegriffe wurden ins Deutsche übernommen und angepasst

Deutsch	Englisch	Französisch
Norm	norm	norme
normal	normal	normal
Normalität	normality	normalité
abnormal	abnormal	anormal

Wie könnte das Wort Inaktivität gebildet worden sein?

Inaktivität → inaktiv → aktiv → akt(ion)/ag(ieren)

analoges deutsches Wort mit ganz anderen Morphemen:

Untätigkeit → untätig → tätig → Tat/tun

Neoklassische Wortbildung

- existiert parallel zur normalen deutschen Wortbildung
- in vielen europäischen Sprachen
- Grund: Latein war lange die Sprache der Wissenschaft. Lateinische (und griechische) Fachbegriffe wurden ins Deutsche übernommen und angepasst

Deutsch	Englisch	Französisch
Norm	norm	norme
normal	normal	normal
Normalität	normality	normalité
abnormal	abnormal	anormal

Wie könnte das Wort Inaktivität gebildet worden sein?

Inaktivität → inaktiv → aktiv → akt(ion)/ag(ieren)

analoges deutsches Wort mit ganz anderen Morphemen:

Untätigkeit → untätig → tätig → Tat/tun

Neoklassische Wortbildung

- existiert parallel zur normalen deutschen Wortbildung
- in vielen europäischen Sprachen
- Grund: Latein war lange die Sprache der Wissenschaft. Lateinische (und griechische) Fachbegriffe wurden ins Deutsche übernommen und angepasst

Deutsch	Englisch	Französisch
Norm	norm	norme
normal	normal	normal
Normalität	normality	normalité
abnormal	abnormal	anormal

Wie könnte das Wort Inaktivität gebildet worden sein?

Inaktivität → inaktiv → aktiv → akt(ion)/ag(ieren)

analoges deutsches Wort mit ganz anderen Morphemen:

Untätigkeit → untätig → tätig → Tat/tun

Neoklassische Wortbildung

- existiert parallel zur normalen deutschen Wortbildung
- in vielen europäischen Sprachen
- Grund: Latein war lange die Sprache der Wissenschaft. Lateinische (und griechische) Fachbegriffe wurden ins Deutsche übernommen und angepasst

Deutsch	Englisch	Französisch
Norm	norm	norme
normal	normal	normal
Normalität	normality	normalité
abnormal	abnormal	anormal

Wie könnte das Wort Inaktivität gebildet worden sein?

Inaktivität → inaktiv → aktiv → akt(ion)/ag(ieren)

analoges deutsches Wort mit ganz anderen Morphemen:

Untätigkeit → untätig → tätig → Tat/tun

Morphologische Prozesse

Komposition

- Bildung einer neuen Wortform aus zwei vorhandenen
  Donau-dampf-schiff-fahrt-s-Kapitän
- Das '-s-' ist ein Fugenmorphem
- Deutsch ist für seine komplexen Komposita bekannt
- Typen deutscher Wortstämme
  - Derivationsstamm (oft mit Umlautung)
    bäuer-lich
  - Kompositionsstämme:

    Bauers-frau, Bauern-krieg, Bauer-beruf
    Baumes-wipfel, Bäume-fällen, baum-lang

    Wie werden diese Stämme gebildet?

  - Einige Derivationsendungen kombinieren mit Kompositionsstämmen:
    taten-los, zahlen-mäßig, damen-haft (vs. däm-lich)

Morphologische Prozesse

Komposition

- Bildung einer neuen Wortform aus zwei vorhandenen
  Donau-dampf-schiff-fahrt-s-Kapitän
- Das '-s-' ist ein Fugenmorphem
- Deutsch ist für seine komplexen Komposita bekannt
- Typen deutscher Wortstämme
  - Derivationsstamm (oft mit Umlautung)
    bäuer-lich
  - Kompositionsstämme:

    Bauers-frau, Bauern-krieg, Bauer-beruf
    Baumes-wipfel, Bäume-fällen, baum-lang

    Wie werden diese Stämme gebildet?

  - Einige Derivationsendungen kombinieren mit Kompositionsstämmen:
    taten-los, zahlen-mäßig, damen-haft (vs. däm-lich)

Morphologische Prozesse

Klitisierung

Ein Klitik ist ein Wort, das mit einem benachbarten Wort verschmolzen (und dabei eventuell reduziert) wurde.

I've	I have (engl.)
C'est	Ce est (franz.)
fermarla	fermar la (ital.)

Morphologische Prozesse

nicht-konkatenative Morphologie

- Hier werden Morpheme nicht einfach aneinandergereiht
- Beispiel: Root-and-Pattern-Morphologie (semitische Sprachen)
  - Wurzel lmd (lernen, Hebräisch)
  - plus das Muster CaCaC (für Aktiv)
  - ergibt das Wort lamad
  - Für die drei C's werden die drei Konsonanten eingesetzt

Mehr zu Morphologie

- Affixe schließen Präfixe, Suffixe, Infixe und Zirkumfixe ein.
  - ge-...-t in getaucht ist ein Zirkumfix (Partizip Perfekt)
  - In abgetaucht stellt ge- ein Infix dar.

- Einige morphologische Prozesse sind nicht mehr produktiv (d.h. sie werden nicht mehr zur Bildung neuer Wörter benutzt)
  - -sam in "einsam", "kleidsam", "arbeitsam"
  - -sal in "Trübsal", "Labsal", "Mühsal"

Flexionsklassen

- Flektierende Wörter kann man in Flexionsklassen einteilen

  Lateinische a-Deklination: casa, casae, casae, casam, casā, ...
  Lateinische o-Deklination: avus, avī, avō, avum, avō, ...
  Lateinische u-Deklination: portus, portūs, portuī, portum, ...

- Alle Wörter einer Flexionsklasse werden nach demselben Schema flektiert.

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'? I

Bedeutung des Präfixes 'a-'? you

Bedeutung des Präfixes 'mi-'? he

Bedeutung des Suffixes '-ban'? past

Bedeutung des Suffixes '-we? present progr.

Bedeutung des Suffixes '-ay? past progr.

Bedeutung des Suffixes '-hi? future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Tiwa-Morphologie (Sprache von Pueblo-Indianern)

temiban	I went
amiban	you went
temiwe	I am going
mimiay	he was going
tewanban	I came
tewanhi	I will come

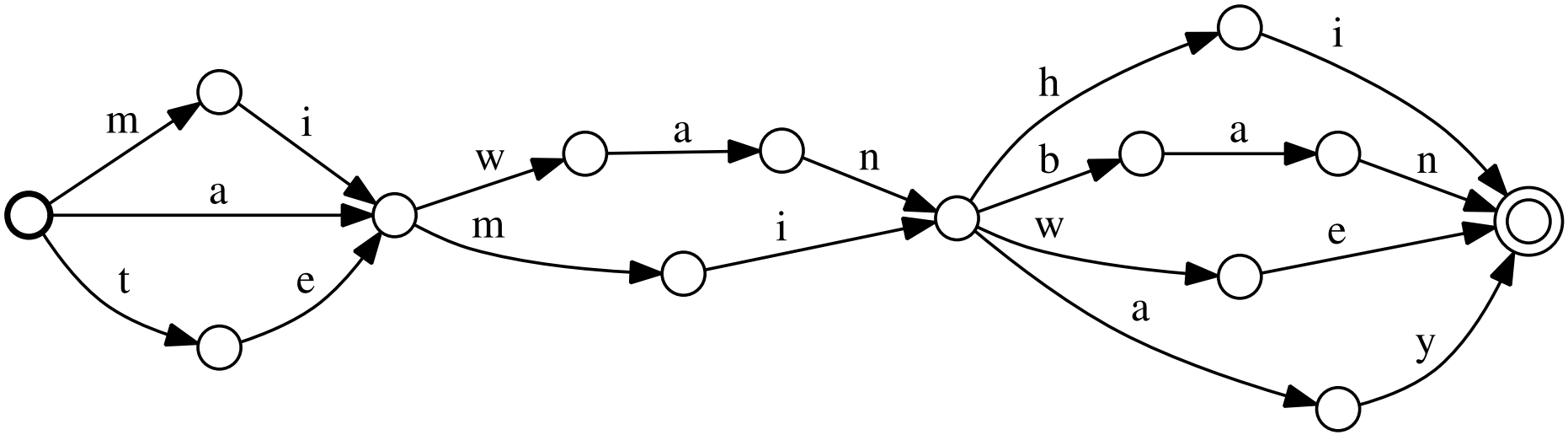
Bedeutung des Präfixes 'te-'?	I
Bedeutung des Präfixes 'a-'?	you
Bedeutung des Präfixes 'mi-'?	he
Bedeutung des Suffixes '-ban'?	past
Bedeutung des Suffixes '-we?	present progr.
Bedeutung des Suffixes '-ay?	past progr.
Bedeutung des Suffixes '-hi?	future

Wie sagt man in Tiwa: "he went" mimiban

regulärer Ausdruck für die Flexion: (te|a|mi) (mi|wan) (ban|we|ay|hi)

Endliche Automaten

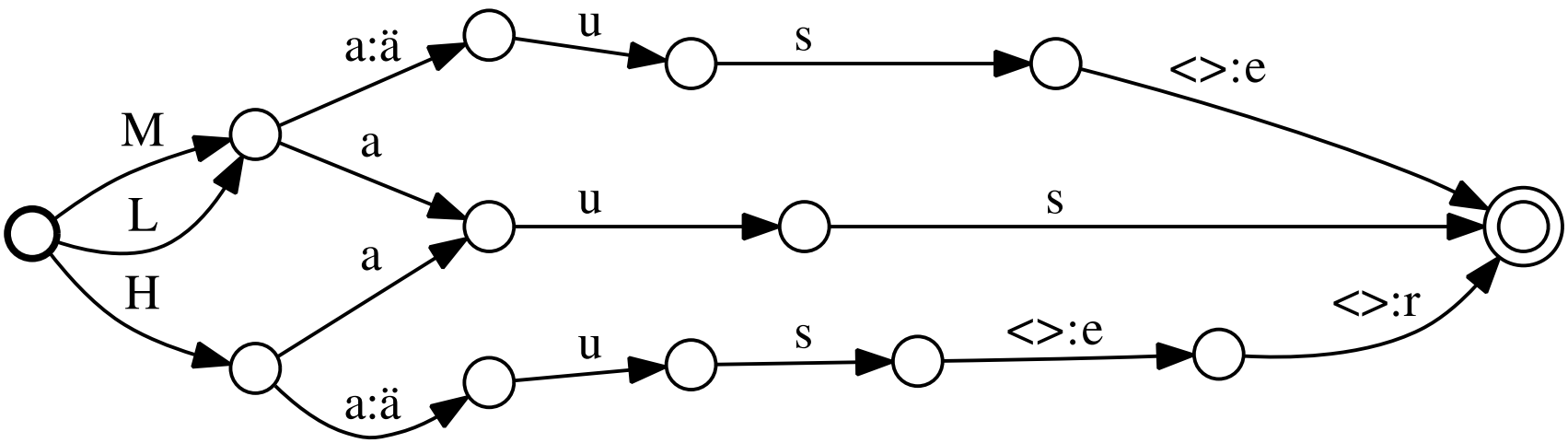
Der reguläre Ausdruck (te|a|mi) (mi|wan) (ban|we|ay|hi) kann in einen endlichen Automaten transformiert werden:



Dieser akzeptiert alle Wörter, die der reguläre Ausdruck bilden kann.

Wir wollen aber nicht nur korrekte Wörter erkennen, sondern auch eine Analyse ausgeben → endliche Transduktoren (finite-state transducer)

Finite State Transducer



Transducer mit den Abbildungen

Haus → Haus
Häuser → Haus
Maus → Maus
Mäuse → Maus
Laus → Laus
Läuse → Laus

Finite State Transducer

- endlicher Automat mit Symbolpaaren an den Übergängen.
- definiert eine Abbildung zwischen zwei regulären Sprachen (reguläre Relation)
- in beide Richtungen (Analyse, Generierung) anwendbar
- Standardmethode zur Implementierung morphologischer Analysewerkzeuge
- FSTs können durch erweiterte reguläre Ausdrücke definiert werden

  (M:M|L:L) a:ä u:u s:s <>:e bildet Mäuse/Läuse auf Maus/Laus ab

  <> ist das leere Symbol

  → Aufgabe 2
  → FST-Programmiersprache

SFST-Werkzeuge

- Programmiersprache für FSTs
- Compiler, der die Programme in Transducer übersetzt
- Programme zur Anwendung von FSTs (Interpreter)
- Beispiel:

```
> echo 'Maus | Laus | (M:M|L:L) a:ä u:u s:s <>:e' > test.fst
> fst-compiler-utf8 test.fst test.a
> fst-mor test.a
analyze> Maus
Maus
analyze> Mäuse
Maus
analyze>          # Leerzeile schaltet auf Generierung um
generate> Maus
Maus
Mäuse
```

Die SFST-Programmiersprache

Reguläre Ausdrücke

- a:b *b* wird als *a* analysiert
- a Abkürzung für a:a
- Konkatenation a b:B c
- Kleene's Operator a*
- Plus Operator b+
- Optionalität <>:e ?
- Klammern (ab)*
- Leerzeichen werden ignoriert.
- Werden die Zeichen :+*?()[ ] und Leerzeichen als solche benötigt, müssen sie mit einem \ gequotet werden.

Die SFST-Programmiersprache

weitere Operatoren

- Disjunktion a:a | a:b | c
- [a-c]:[A-C] kurz für a:A | b:B | c:C
- [a-c] kurz für [a-c]:[a-c]
- {abc}:{AB} kurz für a:A b:B c:<>
- Konjunktion [ab]*abba[ab]* & [ab]*baab[ab]*
- Negation [ab]*abba[ab]* & ![ab]*baab[ab]*
  (äquivalent zu [ab]*abba[ab]* - [ab]*baab[ab]*)

Kommentare werden mit % eingeleitet und enden am Zeilenende.

Variablen beginnen und enden mit $ (bspw. var) und speichern einen regulären Ausdruck (Transducer).

Kompositionsoperator

`$T1$ || $T2$`

- T1 und T2 werden (in Generierungsrichtung) hintereinander ausgeführt.
- Die Ausgabe von T1 bildet die Eingabe von T2
- Der Compiler erzeugt einen Transducer, der dieselbe Abbildung in einem Schritt ausführt.

Beispiel: `[a-z]:[A-Z]* || [A-Z]:[a-z]*`

Was macht dieser Transducer?

Der Transducer ist identisch zu `[a-z]*`

Kompositionsoperator

`$T1$ || $T2$`

- T1 und T2 werden (in Generierungsrichtung) hintereinander ausgeführt.
- Die Ausgabe von T1 bildet die Eingabe von T2
- Der Compiler erzeugt einen Transducer, der dieselbe Abbildung in einem Schritt ausführt.

Beispiel: `[a-z]:[A-Z]* || [A-Z]:[a-z]*`

Was macht dieser Transducer?

Der Transducer ist identisch zu `[a-z]*`

Das Alphabet

Der Befehl `ALPHABET = [a-z]:[A-Z] [a-z]:[a-z]` definiert ein Alphabet.

Auf der rechten Seite kann ein beliebiger Transducerausdruck stehen.

Das Alphabet umfasst alle Symbolpaare, die an Übergängen des Transducers auftauchen.

Das Wildcardsymbol "." steht für die Disjunktion aller Symbolpaare im Alphabet.

"a:." steht für die Disjunktion aller Symbolpaare im Alphabet mit "a" auf der linken Seite.

Was macht der Transducer ".*" bei obiger Definition des Alphabetes?

Das Alphabet

Der Befehl `ALPHABET = [a-z]:[A-Z] [a-z]:[a-z]` definiert ein Alphabet.

Auf der rechten Seite kann ein beliebiger Transducerausdruck stehen.

Das Alphabet umfasst alle Symbolpaare, die an Übergängen des Transducers auftauchen.

Das Wildcardsymbol "." steht für die Disjunktion aller Symbolpaare im Alphabet.

"a:." steht für die Disjunktion aller Symbolpaare im Alphabet mit "a" auf der linken Seite.

Was macht der Transducer ".*" bei obiger Definition des Alphabetes?

SFST-Programme

Programmstruktur

Ein SFST-Programm besteht aus einer Folge von Variablen-Definitionen, Alphabet-Definitionen und anderen Befehlen gefolgt von einem einzigen Ergebnisausdruck, dessen Wert nach der Kompilierung in der Ausgabedatei gespeichert wird.

Es werden aus einfacheren Transducern (regulären Ausdrücken) immer komplexere Transducer aufgebaut, bis der gewünschte Ergebnis-Transducer fertiggestellt ist.

SFST-Beispiele

```
ALPHABET = [a-zA-Z]
```

```
[a-zA-Z]:[A-Za-z]+
```

Welche “Analyse” wird ausgegeben für “LaTeX”?

```
[ab]* - .*ba.*
```

Welche Strings akzeptiert dieser Automat?

```
.* {ll}:{LL}? .*
```

“Analyse” von “Hallo”?

“Analyse” von “HaLLo”?

```
{Hallo}:{<>Hallo}
```

“Analyse” von “Hallo”?

```
{Hallo}:{<>Hallo} & Hallo
```

Ausgabe für “Hallo”?

SFST-Beispiele

```
ALPHABET = [a-zA-Z]
```

```
[a-zA-Z]:[A-Za-z]+
```

Welche “Analyse” wird ausgegeben für “LaTeX”?

```
[ab]* - .*ba.*
```

Welche Strings akzeptiert dieser Automat?

```
.* {ll}:{LL}? .*
```

“Analyse” von “Hallo”?

“Analyse” von “HaLLo”?

```
{Hallo}:{<>Hallo}
```

“Analyse” von “Hallo”?

```
{Hallo}:{<>Hallo} & Hallo
```

Ausgabe für “Hallo”?

SFST-Beispiele

```
ALPHABET = [a-zA-Z]
```

```
[a-zA-Z]:[A-Za-z]+
```

Welche “Analyse” wird ausgegeben für “LaTeX”?

```
[ab]* - .*ba.*
```

Welche Strings akzeptiert dieser Automat?

```
.* {ll}:{LL}? .*
```

“Analyse” von “Hallo”?

“Analyse” von “HaLLo”?

```
{Hallo}:{<>Hallo}
```

“Analyse” von “Hallo”?

```
{Hallo}:{<>Hallo} & Hallo
```

Ausgabe für “Hallo”?

SFST-Beispiele

```
ALPHABET = [a-zA-Z]
```

```
[a-zA-Z]:[A-Za-z]+
```

Welche “Analyse” wird ausgegeben für “LaTeX”?

```
[ab]* - .*ba.*
```

Welche Strings akzeptiert dieser Automat?

```
.* {ll}:{LL}? .*
```

“Analyse” von “Hallo”?

“Analyse” von “HaLLo”?

```
{Hallo}:{<>Hallo}
```

“Analyse” von “Hallo”?

```
{Hallo}:{<>Hallo} & Hallo
```

Ausgabe für “Hallo”?

SFST-Beispiele

```
ALPHABET = [a-zA-Z]
```

```
[a-zA-Z]:[A-Za-z]+
```

Welche "Analyse" wird ausgegeben für "LaTeX"?

```
[ab]* - .*ba.*
```

Welche Strings akzeptiert dieser Automat?

```
.* {ll}:{LL}? .*
```

"Analyse" von "Hallo"?
"Analyse" von "HaLLo"?

```
{Hallo}:{<>Hallo}
```

"Analyse" von "Hallo"?

```
{Hallo}:{<>Hallo} & Hallo
```

Ausgabe für "Hallo"?

SFST-Beispiele

```
ALPHABET = [a-zA-Z]
```

```
[a-zA-Z]:[A-Za-z]+
```

Welche "Analyse" wird ausgegeben für "LaTeX"?

```
[ab]* - .*ba.*
```

Welche Strings akzeptiert dieser Automat?

```
.* {ll}:{LL}? .*
```

"Analyse" von "Hallo"?
"Analyse" von "HaLLo"?

```
{Hallo}:{<>Hallo}
```

"Analyse" von "Hallo"?

```
{Hallo}:{<>Hallo} & Hallo
```

Ausgabe für "Hallo"?

Tokenisierung mit FSTs

Einfacher Beispieltransducer, der Textstrings tokenisiert.

Definition von Variablen

```
$Letter$ = [A-ZÄÖÜa-zäöüß]
$leftPunc$ = ["]
$rightPunc$ = [.,;:!?")]
```

Satzzeichen und Klammern an den Worträndern abtrennen

```
$X$ = ($leftPunc$ \ :<>)* $Letter$* (\ :<> $rightPunc$)*
```

Wie sieht der Transducer in X aus?

Ergebnisausdruck

```
($X$ \ +)* $X$
```

Tokenisierung mit FSTs

Einfacher Beispieltransducer, der Textstrings tokenisiert.

Definition von Variablen

```
$Letter$ = [A-ZÄÖÜa-zäöüß]
$leftPunc$ = ["]
$rightPunc$ = [.,;:!?")]
```

Satzzeichen und Klammern an den Worträndern abtrennen

```
$X$ = ($leftPunc$ \ :<>)* $Letter$* (\ :<> $rightPunc$)*
```

Wie sieht der Transducer in X aus?

Ergebnisausdruck

```
($X$ \ +)* $X$
```

Tokenisierung mit FSTs

Anwendung des Transducers

```
> fst-compiler-utf8 tokenize.fst tokenize.a
> fst-mor tokenize.a
analyze> Er kam, sah, und siegte.
Er kam , sah , und siegte .
```

Erkennung von Abkürzungen

Definition von Variablen

```
...
$all$ = $Letter$ $leftPunc$ $rightPunc$ [\ ]
```

Satzzeichen und Klammern an den Worträndern abtrennen

```
$X$ = ($leftPunc$ \ :<>)* $Letter$* (\ :<> $rightPunc$)*
$X$ = ($X$ \ +)* $X$
```

Abkürzungen wieder zusammenfügen

```
ALPHABET = $all$
$R$ =  (<>:\ ) _-> ((Fr|Hr|Prof|Dr|etc|usw) __ \.)

$R$ || $X$
```

```
analyze> "Heureka!", sagte Hr. Müller.

" Heureka ! " , sagte Hr. Müller .
```

Ersetzungsregeln

`(b:B) ^-> (a __ c)`	Ersetze b zwischen a und c durch B an der Oberfläche
`(b:B) _-> (A __ C)`	Ersetze B zwischen A und C durch b in der Analyse

Bei leerem Kontext wird immer ersetzt:

`(b:B) ^-> ()` `(b:B) _-> ()`

allgemeine Syntax: Ersetzung ^-> (linker Kontext __ rechter Kontext)

Ersetzung ist ein beliebiger Transducer

linker/rechter Kontext muss ein Automat sein (d.h. darf kein ':' enthalten)

Weitere Operatoren

Der Operator `^$A$` macht die Analysesymbole im Transducer `$A$` identisch zu den Oberflächensymbolen (also `^a:b = b:b`)

Der Operator `_$A$` macht die Oberflächensymbole identisch zu den Analysesymbolen (also `_a:b = a:a`)

Der Operator `^_$A$` vertauscht die Oberflächen- und Analysesymbole (also `^_{ab}:{AB} = {AB}:{ab}`)

Der Operator `"Datei"` liest den Inhalt von "Datei" und bildet einen Transducer durch Disjunktion der eingelesenen Zeilen. In der eingelesenen Datei werden nur die Symbole `< > : \` als Operatoren interpretiert.

Der Operator `"<Datei>"` liest einen bereits kompilierten Transducer aus "Datei".

Zeichenketten der Form `<...>` werden als atomare Symbole behandelt wie einzelne Buchstaben.

Morphologie

Lexikoneinträge aus Datei einlesen, ein Wort pro Zeile

```
$nouns$ = "nouns"
```

Endungen und Analysen hinzufügen

```
$MORPH$ = $nouns$ <N> <sg>:<> | $nouns$ <N> <pl>:s
```

morphophonologische Regeln definieren (story<N>s → storie<N>s)

```
ALPHABET = [A-Za-z<N>]

$R1$ = (y:{ie}) ^-> ([^aeiou]__ <N>s)
$R2$ = (<N>:<>) ^-> ()
```

Regeln anwenden

```
$MORPH$ || $R1$ || $R2$
```

Ausführen der Morphologie

```
> fst-compiler-utf8 morph.fst morph.a
> fst-mor morph.a
analyze> lobbies
lobby<N><pl>
analyze> stores
store<N><pl>
analyze> houses
house<N><pl>
analyze>
generate> house<N><pl>
houses
generate> house<N><sg>
house
generate> q
```

Einlesen von Lexikondateien

```
$nouns$ = "nouns"
$verbs$ = "verbs"
$present-verbs$ = "present-verbs"
$past-verbs$ = "past-verbs"
$part-verbs$ = "part-verbs"
$adjectives$ = "adjectives"
$gradable-adjectives$ = "gradable-adjectives"

$MORPH$ = $nouns$ <N> (<sg>:<> | <pl>:s)

...
```

Anfügen der Analysen und Flexionsendungen

```
...
$MORPH$ = $MORPH$ |\
   $verbs$ <V> {<inf>}:{} |\
   $verbs$ <V> {<pres><n3s>}:{} |\
   $verbs$ <V> {<pres><3s>}:{s} |\
   $verbs$ <V> {<gerund>}:{ing} |\
   $verbs$ <V> {<past>}:{ed} |\
   $verbs$ <V> {<part>}:{ed} |\
   $present-verbs$ <V> {<inf>}:{} |\
   $present-verbs$ <V> {<pres><n3s>}:{} |\
   $present-verbs$ <V> {<pres><3s>}:{s} |\
   $present-verbs$ <V> {<gerund>}:{ing} |\
   $past-verbs$ <V> {<past>}:{} |\
   $part-verbs$ <V> {<part>}:{}

$MORPH$ = $MORPH$ |\
   $adjectives$ <ADJ> {<pos>}:{} |\
   $gradable-adjectives$ <ADJ> {<pos>}:{} |\
   $gradable-adjectives$ <ADJ> {<comp>}:{er} |\
   $gradable-adjectives$ <ADJ> {<sup>}:{est}
...
```

Morphophonologische Regeln

Folgende Phänomene sollen behandelt werden

story+s → stories
happy+er → happier
late+est → latest

Wie könnten die Regeln dafür aussehen?

```
...

ALPHABET = ^$MORPH$

$R1$ = (y:{ie}) ^-> ([^aeiou] __ [<N><V>]s)
$R2$ = (y:i) ^-> (__ <ADJ>e)
$R3$ = (e:<>) ^-> (__ [<ADJ><V>][ei])
$R4$ = ([<N><V><ADJ>]:<>) ^-> ()

$MORPH$ || $R1$ || $R2$ || $R3$ || $R4$
```

Morphophonologische Regeln

Folgende Phänomene sollen behandelt werden

story+s → stories
happy+er → happier
late+est → latest

Wie könnten die Regeln dafür aussehen?

```
...

ALPHABET = ^$MORPH$

$R1$ = (y:{ie}) ^-> ([^aeiou] __ [<N><V>]s)
$R2$ = (y:i) ^-> (__ <ADJ>e)
$R3$ = (e:<>) ^-> (__ [<ADJ><V>][ei])
$R4$ = ([<N><V><ADJ>]:<>) ^-> ()

$MORPH$ || $R1$ || $R2$ || $R3$ || $R4$
```

Morphophonologische Regeln

Bei der Bildung des englischen Komparativs oder Superlativs wird der letzte Konsonant des Adjektivs verdoppelt, wenn

- das Adjektiv nur eine Silbe umfasst und
- auf Konsonant-Vokal-Konsonant endet und
- der letzte Konsonant nicht "y" oder "w" ist

big	bigger	biggest
fat	fatter	fattest
thin	thinner	thinnest
grey	greyer	greyest
slow	slower	slowest
plain	plainer	plainest

Wie könnte eine Regel dafür aussehen?

```
({g}:{gg}|{t}:{tt}|{n}:{nn}) ^-> (<B> $cons$+ $vowel$ __ <ADJ>e)
```

<B> ist hier eine Grenzmarkierung, die eingefügt werden muss, um den Wortanfang matchen zu können. Siehe dazu auch Übungsaufgabe 2.

Morphophonologische Regeln

Bei der Bildung des englischen Komparativs oder Superlativs wird der letzte Konsonant des Adjektivs verdoppelt, wenn

- das Adjektiv nur eine Silbe umfasst und
- auf Konsonant-Vokal-Konsonant endet und
- der letzte Konsonant nicht "y" oder "w" ist

big	bigger	biggest
fat	fatter	fattest
thin	thinner	thinnest
grey	greyer	greyest
slow	slower	slowest
plain	plainer	plainest

Wie könnte eine Regel dafür aussehen?

```
({g}:{gg}|{t}:{tt}|{n}:{nn}) ^-> (<B> $cons$+ $vowel$ __ <ADJ>e)
```

<B> ist hier eine Grenzmarkierung, die eingefügt werden muss, um den Wortanfang matchen zu können. Siehe dazu auch Übungsaufgabe 2.

Kopier-Operationen

```
({g}:{gg}|{t}:{tt}|{n}:{nn})
```

Diese Ausdruck zum Kopieren eines Buchstabens wird sehr lang, wenn er alle Konsonanten umfassen soll. Die Lösung sind Kopiervariablen.

Variablen für Symbolmengen:

```
#vowel# = aeiou
#cons# = bcdfghjklmnpqrstvx
$vowel$ = [#vowel#]
$cons$ = [#cons#]
```

Kopiervariable:

```
#=c# = #cons#

{[#=c#]}:{[#=c#][#=c#]} ^-> (<B> $cons$+ $vowel$ __ <ADJ>e)
```

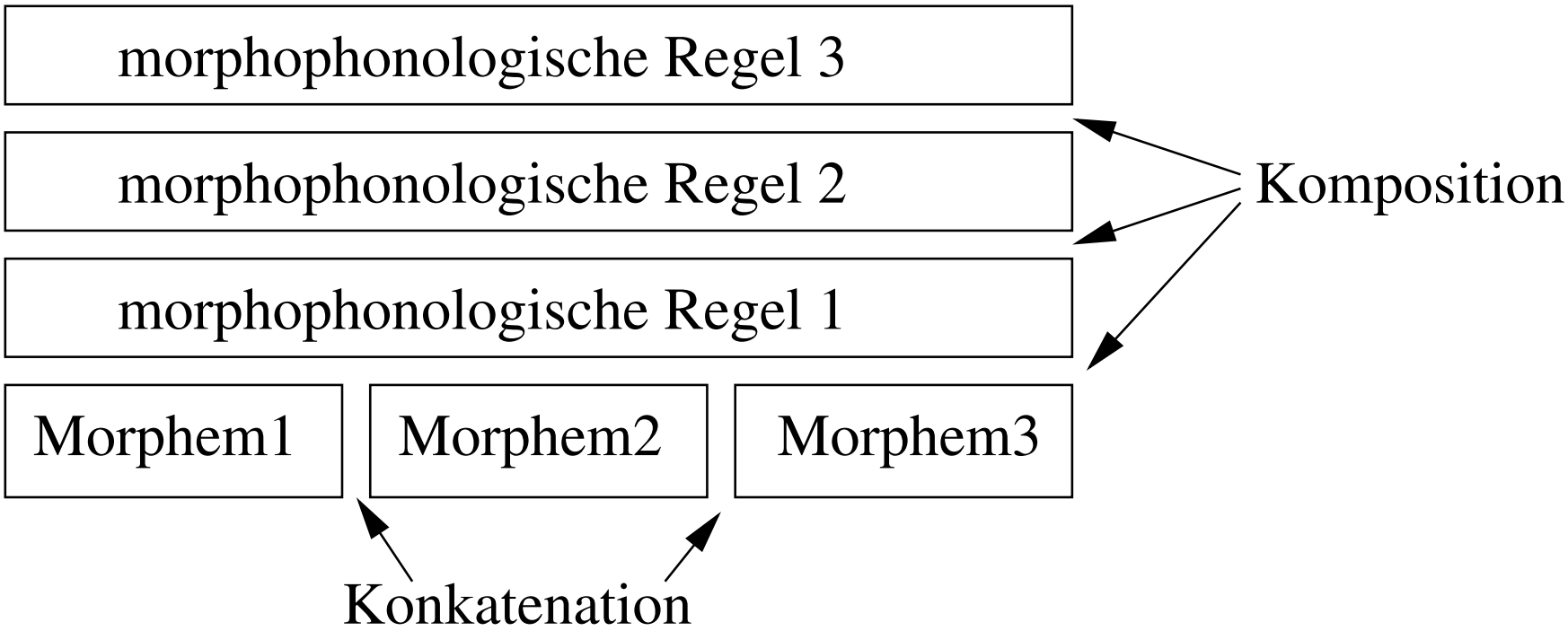
Entwicklung einer Morphologie

Vorgehensweise

- zu behandelnde Phänomene festlegen
  - Flexion, Derivation, Komposition?
  - welche Wortarten?
  - unregelmäßig gebildete Formen?
  - welche morphophonologische Regeln?
  - weitere Phänomene wie Klitika?
- Stammlexika anlegen
- Lexika einlesen
- ggf. Derivationsendungen hinzufügen
- ggf. Kompositums-Erstglieder hinzufügen
- Flexionsendungen hinzufügen
- morphophonologische Regeln anwenden

Entwicklung einer Morphologie

- Konkatenation der Morpheme
- Anwendung der morphophonologischen Regeln mit Komposition



Debugging

- Testwörter ausprobieren
- Zwischenergebnisse im Programm ausgeben mit dem Befehl
  `$var$ >> "tmp.a"`
  Dann den ausgegebenen Transducer untersuchen mit
  `> fst-generate tmp.a`
  `> fst-generate -l tmp.a` nur die Strings der Analyseebene
  `> fst-generate -u tmp.a` nur die Strings der Oberflächenebene
  Mit *grep* die interessanten String herausfiltern
- Bei Fehlersuche die Lexika auf genau die relevanten Einträge reduzieren
  $\Rightarrow$ schnellere Kompilierung, einfachere Fehlersuche

Beispiel

```
> cat test.fst
{Hut}:{Hüte}
> fst-compiler-utf8 test.fst test.a
test.fst: 2
> fst-print test.a
0   1   H   H
1   2   u   ü
2   3   t   t
3   4   <>  e
4
> fst-generate test.a
Hu:üt<>:e
```

Aufgabe bis morgen

- SFST-Tutorial auf der Kursseite durcharbeiten
- Gedanken zur neuen Übungsaufgabe machen

Textklassifikation

- Wozu dient Textklassifikation?
- im Folgenden einige Beispiele

Spam-Erkennung

Wie kann man Spam von normaler Mail unterscheiden?

Subject: Great news!
From: XXY@abc.com
To: undisclosed-recipients;
Congratulations!
You have been selected to be the winner of XXX. Click on the following link to login for more information about the exciting price.
http://www.123contact.com/contact-form-1234.html
Regards,
XYZ

Spam-Erkennung

Wie kann man Spam von normaler Mail unterscheiden?

Subject: Great news!
From: XXY@abc.com
To: undisclosed-recipients;
Congratulations!
You have been selected to be the winner of XXX. Click on the following link to login for more information about the exciting price.
http://www.123contact.com/contact-form-1234.html
Regards,
XYZ

Filmbewertungen

Wurde der Film positiv oder negativ rezensiert?

1. Dieser Film ist großartig!
2. Das ist der langweiligste Film, den ich je gesehen habe.
3. Die Leistung der Schauspieler ist nicht überzeugend.
4. Der Film ist nicht schlecht.
5. Wenn Sie ein absoluter Fan von Till Schweiger sind, gefällt Ihnen dieser Film vielleicht.

Filmbewertungen

Wurde der Film positiv oder negativ rezensiert?

1. Dieser Film ist großartig! ☺
2. Das ist der langweiligste Film, den ich je gesehen habe. ☹
3. Die Leistung der Schauspieler ist nicht überzeugend. ☹
4. Der Film ist nicht schlecht. ☺?
5. Wenn Sie ein absoluter Fan von Till Schweiger sind, gefällt Ihnen dieser Film vielleicht. ☹

Filmbewertungen

Wurde der Film positiv oder negativ rezensiert?

1. Dieser Film ist großartig! ☺
2. Das ist der langweiligste Film, den ich je gesehen habe. ☹
3. Die Leistung der Schauspieler ist nicht überzeugend. ☹
4. Der Film ist nicht schlecht. ☺?
5. Wenn Sie ein absoluter Fan von Till Schweiger sind, gefällt Ihnen dieser Film vielleicht. ☹

Filmbewertungen

Wurde der Film positiv oder negativ rezensiert?

1. Dieser Film ist großartig! ☺
2. Das ist der langweiligste Film, den ich je gesehen habe. ☹
3. Die Leistung der Schauspieler ist nicht überzeugend. ☹
4. Der Film ist nicht schlecht. ☺?
5. Wenn Sie ein absoluter Fan von Till Schweiger sind, gefällt Ihnen dieser Film vielleicht. ☹

Filmbewertungen

Wurde der Film positiv oder negativ rezensiert?

1. Dieser Film ist großartig! ☺
2. Das ist der langweiligste Film, den ich je gesehen habe. ☹
3. Die Leistung der Schauspieler ist nicht überzeugend. ☹
4. Der Film ist nicht schlecht. ☺?
5. Wenn Sie ein absoluter Fan von Till Schweiger sind, gefällt Ihnen dieser Film vielleicht. ☹

Filmbewertungen

Wurde der Film positiv oder negativ rezensiert?

1. Dieser Film ist großartig! ☺
2. Das ist der langweiligste Film, den ich je gesehen habe. ☹
3. Die Leistung der Schauspieler ist nicht überzeugend. ☹
4. Der Film ist nicht schlecht. ☺?
5. Wenn Sie ein absoluter Fan von Till Schweiger sind, gefällt Ihnen dieser Film vielleicht. ☹

Textklassifikationsmethoden

- handgeschriebene Regeln
  - Regeln auf Basis der Wörter und anderer Merkmale (z.B. Listen bekannter Spam-Adressen)
  - Gute Genauigkeit, aber hoher Aufwand
  - Regeln müssen aufwendig angepasst werden, wenn sich die Daten ändern (z.B. weil Spammer gelernt haben, die Regeln auszutricksen)
- maschinelles Lernen
  - Der Computer lernt die Klassifikation selbst aus Beispielen, deren Klasse bekannt ist
  - ideal, wenn solche Beispiele bereits vorhanden sind
  - Eine Methode für die Textklassifikation ist → Naïve Bayes

Textklassifikationsmethoden

- handgeschriebene Regeln
  - Regeln auf Basis der Wörter und anderer Merkmale (z.B. Listen bekannter Spam-Adressen)
  - Gute Genauigkeit, aber hoher Aufwand
  - Regeln müssen aufwendig angepasst werden, wenn sich die Daten ändern (z.B. weil Spammer gelernt haben, die Regeln auszutricksen)
- maschinelles Lernen
  - Der Computer lernt die Klassifikation selbst aus Beispielen, deren Klasse bekannt ist
  - ideal, wenn solche Beispiele bereits vorhanden sind
  - Eine Methode für die Textklassifikation ist → Naïve Bayes

Textklassifikation mit Naïve Bayes

Grundidee:

- Wörter, die häufig in Texten einer bestimmten Klasse auftauchen, sind ein Indiz für diese Klasse.
- Jedes Wort wird einzeln betrachtet.
- Die Reihenfolge der Wörter spielt keine Rolle
  $\Rightarrow$ "Bag of Words"-Modell

Textklassifikation mit Naïve Bayes

Wir wollen die wahrscheinlichste Klasse $\hat{c}$ des Dokumentes d bestimmen:

$$\hat{c} = \arg\max_c p(c|d)$$

Die bed. Wk. $p(c|d)$ kann nicht direkt geschätzt werden (zu viele Parameter)

Wir wandeln den Ausdruck um (gemäß der Def. von bed. Wken)

$$\hat{c} = \arg\max_c \frac{p(c,d)}{p(d)}$$

Die positive Konstante $p(d)$ hat keinen Einfluß auf das Ergebnis der Maximierung.

$$\hat{c} = \arg\max_c p(c,d)$$

Das gemeinsame Modell von c und d können wir umschreiben zu:

$$\hat{c} = \arg\max_c p(c)\ p(d|c)$$

Parameterschätzung

$$\hat{c} = \arg\max_c p(c)\ p(d|c)$$

$p(c)$ kann nach folgender Formel aus den Klassenhäufigkeiten $f(c)$ geschätzt werden:

$$p(c) = \frac{f(c)}{\sum_{c'} f(c')}$$

Aber wie bekommt man $p(d|c)$?

Naïve Bayes

d besteht aus der Folge der Wörter $w_1, ..., w_n$

Wir nehmen an, dass die Reihenfolge der Wörter egal ist, und dass die Wörter statistisch voneinander unabhängig sind, wenn c bekannt ist, also $p(w, w'|c) = p(w|c)p(w'|c)$

Somit gilt:

$$p(d|c) = p(w_1, ..., w_n|c) = \prod_{i=1}^{n+1} p(w_i|c)$$

Wir fügen hier ein Endesymbol w_{n+1} hinzu, damit wir eine Wahrscheinlichkeitsverteilung über Wortfolgen beliebiger Länge erhalten.

Zusammengefasst ergibt sich das Naïve Bayes-Modell:

$$\hat{c} = \arg\max_c p(c) \prod_{i=1}^{n+1} p(w_i|c)$$

Naïve Bayes

modellierter Prozess:

- zufällige Wahl eines Dokumenttyps (Spam, Ham)
- Wiederhole
  - zufällige Wahl eines Wortes gegeben die Dokumentklasse
  - bis ein Endetoken gewählt wurde

Parameterschätzung

Die Wortwahrscheinlichkeiten können mit relativen Häufigkeiten geschätzt werden

$$p(w|c) = \frac{f(w,c)}{\sum_{w'} f(w',c)}$$

Wenn ein Wort nicht in Dokumenten der Klasse c aufgetaucht ist, wird $p(w|c) = 0$ und damit auch der ganze Ausdruck des Naïve Bayes-Modelles.

$\rightarrow$ Beispiel

Naïve Bayes-Beispiel

	Dok	Wörter	Klasse
Training	1	München Bremen Deutschland	de
	2	München München Deutschland	de
	3	München Bremen	de
	4	Frankreich Lyon Lyon München	fr
Test	5	München München Lyon	?

$p(de) = 3/4 \quad p(fr) = 1/4$

$p(\text{München}|de) = 4/8 = 1/2 \qquad p(\text{München}|fr) = 1/4$
$p(\text{Lyon}|de) = 0/8 = 0 \qquad p(\text{Lyon}|fr) = 2/4 = 1/2$

fr: $1/4 * (1/4 * 1/4 * 1/2) = 1/128$
de: $3/4 * (1/2 * 1/2 * 0) = 0$

$\rightarrow$ Parameterglättung, damit alle Wahrscheinlichkeiten größer 0 sind.

Naïve Bayes-Beispiel

	Dok	Wörter	Klasse
Training	1	München Bremen Deutschland	de
	2	München München Deutschland	de
	3	München Bremen	de
	4	Frankreich Lyon Lyon München	fr
Test	5	München München Lyon	?

$p(de) = 3/4 \quad p(fr) = 1/4$

$p(\text{München}|de) = 4/8 = 1/2 \quad p(\text{München}|fr) = 1/4$

$p(\text{Lyon}|de) = 0/8 = 0 \quad p(\text{Lyon}|fr) = 2/4 = 1/2$

fr: $1/4 * (1/4 * 1/4 * 1/2) = 1/128$

de: $3/4 * (1/2 * 1/2 * 0) = 0$

$\rightarrow$ Parameterglättung, damit alle Wahrscheinlichkeiten größer 0 sind.

Naïve Bayes-Beispiel

	Dok	Wörter	Klasse
Training	1	München Bremen Deutschland	de
	2	München München Deutschland	de
	3	München Bremen	de
	4	Frankreich Lyon Lyon München	fr
Test	5	München München Lyon	?

$p(de) = 3/4 \quad p(fr) = 1/4$

$p(\text{München}|de) = 4/8 = 1/2 \qquad p(\text{München}|fr) = 1/4$
$p(\text{Lyon}|de) = 0/8 = 0 \qquad p(\text{Lyon}|fr) = 2/4 = 1/2$

fr: $1/4 * (1/4 * 1/4 * 1/2) = 1/128$
de: $3/4 * (1/2 * 1/2 * 0) = 0$

$\rightarrow$ Parameterglättung, damit alle Wahrscheinlichkeiten größer 0 sind.

Naïve Bayes-Beispiel

	Dok	Wörter	Klasse
Training	1	München Bremen Deutschland	de
	2	München München Deutschland	de
	3	München Bremen	de
	4	Frankreich Lyon Lyon München	fr
Test	5	München München Lyon	?

$p(de) = 3/4 \quad p(fr) = 1/4$

$p(\text{München}|de) = 4/8 = 1/2 \qquad p(\text{München}|fr) = 1/4$
$p(\text{Lyon}|de) = 0/8 = 0 \qquad p(\text{Lyon}|fr) = 2/4 = 1/2$

fr: $1/4 * (1/4 * 1/4 * 1/2) = 1/128$
de: $3/4 * (1/2 * 1/2 * 0) = 0$

→ Parameterglättung, damit alle Wahrscheinlichkeiten größer 0 sind.

Backoff-Glättung

Wir berechnen relative Häufigkeiten, wobei von den positiven Häufigkeiten ein kleiner Betrag δ (Discount) abgezogen wird:

$$r(w|c) = \frac{max(0, f(w,c) - \delta)}{\sum_{w'} f(w', c)}$$

Nullhäufigkeiten bleiben wegen der max-Operation unverändert.

Durch das Discounting wird Wahrscheinlichkeitsmasse frei, die über alle Wörter proportional zu einer Backoff-Verteilung $p(w)$ verteilt wird:

$$p(w|c) = r(w|c) + \alpha(c)p(w)$$

Der Backoff-Faktor $\alpha(c)$ wird so definiert, dass $\sum_w p(w|c) = 1$:

$$\alpha(c) = 1 - \sum_{w'} r(w'|c)$$

Discount

Nach Kneser/Essen/Ney kann der Discount folgendermaßen definiert werden:

$$\delta = \frac{N_1}{N_1 + 2N_2}$$

N_1 ist die Zahl der Paare (w,c) mit $f(w, c) = 1$

N_2 ist die Zahl der Paare (w,c) mit $f(w, c) = 2$

Wie wird die Backoff-Wahrscheinlichkeitsverteilung $p(w)$ definiert?

Backoff-Verteilung

Als Backoff-Verteilung nehmen wir die Apriori-Wahrscheinlichkeiten der Wörter

$$p(w) = \frac{f(w)}{\sum_{w'} f(w')}$$

mit $f(w) = \sum_c f(w, c)$

$\rightarrow$ Aufgabe 3

Nachteile von Naïve Bayes

- Wörter sind nicht immer statistisch unabhängig
  - Beispiel: Wenn ein Text das Wort Francisco enthält, enthält er meist auch das Wort San. Daher

$$p(\text{San Francisco}|c) \neq p(\text{San}|c)\ p(\text{Francisco}|c)$$

    sondern eher

$$p(\text{San Francisco}|c) \approx p(\text{Francisco}|c)$$

- Es ist schwierig, beliebige weitere Merkmale in das Modell zu integrieren z.B.
  - Wortpräfixe
  - ob ein Wort in einem Eigennamen-Lexikon enthalten ist
  - an welcher Position im Artikel sich ein Wort befindet

→ Diskriminative Modelle vermeiden diese Probleme

Nachteile von Naïve Bayes

- Wörter sind nicht immer statistisch unabhängig
  - Beispiel: Wenn ein Text das Wort Francisco enthält, enthält er meist auch das Wort San. Daher

$$p(\text{San Francisco}|c) \neq p(\text{San}|c)\ p(\text{Francisco}|c)$$

    sondern eher

$$p(\text{San Francisco}|c) \approx p(\text{Francisco}|c)$$

- Es ist schwierig, beliebige weitere Merkmale in das Modell zu integrieren z.B.
  - Wortpräfixe
  - ob ein Wort in einem Eigennamen-Lexikon enthalten ist
  - an welcher Position im Artikel sich ein Wort befindet

$\rightarrow$ Diskriminative Modelle vermeiden diese Probleme

Diskriminative Modelle

Generative Modelle (wie Naïve Bayes) definieren eine gemeinsame Wahrscheinlichkeit von Klasse c und Dokument d:

$$p(c, d)$$

Diskriminative Modelle definieren eine bedingte Wahrscheinlichkeit der Klasse c gegeben Dokument d:

$$p(c|d)$$

Beispiele von diskriminativen Modellen

- log-lineare Modelle (Maximum-Entropy-Modelle)
- Conditional Random Fields

Diskriminative Modelle

Generative Modelle (wie Naïve Bayes) definieren eine gemeinsame Wahrscheinlichkeit von Klasse c und Dokument d:

$$p(c, d)$$

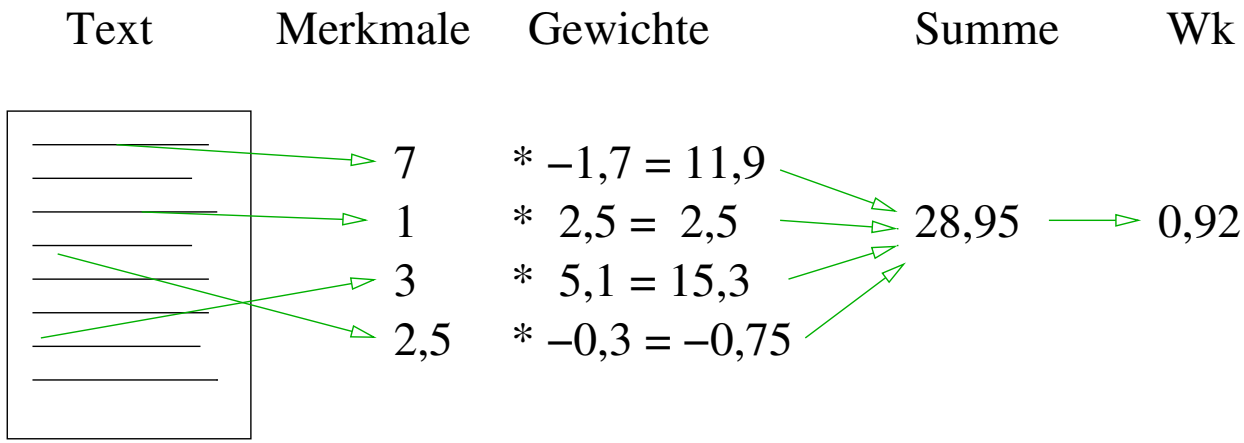
Diskriminative Modelle definieren eine bedingte Wahrscheinlichkeit der Klasse c gegeben Dokument d:

$$p(c|d)$$

Beispiele von diskriminativen Modellen

- log-lineare Modelle (Maximum-Entropy-Modelle)
- Conditional Random Fields

Log-lineare Modelle



Grundidee:

- Manuell erstellte Merkmalsfunktionen extrahieren relevante Information aus dem Dokument.
- Jeder Merkmalswert wird mit einem Gewicht multipliziert, das ausdrückt, ob das Merkmal auf eine bestimmte Klasse hindeutet.
- Die gewichteten Merkmalswerte werden aufsummiert und in Wahrscheinlichkeiten transformiert.

Merkmale

Log-lineare Modelle definieren eine Menge von Merkmalsfunktionen $f_1(c,d), ..., f_M(c,d)$

- binäre Merkmale

$$f_1(c,d) = \begin{cases} 1 & \text{falls Textlänge} \in [20, ..., 40) \text{ und } c = \text{negativ} \\ 0 & \textit{sonst} \end{cases}$$

- numerische Merkmale

$$f_2(c,d) = \begin{cases} n & \text{falls ,,toll'' n-mal auftaucht und } c = \text{positiv} \\ 0 & \text{falls } c \neq \text{positiv} \end{cases}$$

Merkmale

Beispieltext:

Das ist ein spannender Film aber die Handlung des Filmes ist sehr klischeehaft.

extrahierte Merkmale (nach Stoppwortentfernung und Lemmatisierung)

Klasse: positiv	
positiv, spannend	1
positiv, Film	2
positiv, Handlung	1
positiv, klischeehaft	1

Klasse: negativ	
negativ, spannend	1
negativ, Film	2
negativ, Handlung	1
negativ, klischeehaft	1

Gewichte

Jedem Merkmal $f_i(c, d)$ wird ein Gewicht θ_i zugeordnet

- $\theta_i > 0$: das Merkmal $f_i(c, d)$ deutet auf die Klasse c hin
- $\theta_i < 0$: das Merkmal $f_i(c, d)$ deutet auf eine andere Klasse als c hin

Die Gewichte sind bel. reelle Zahlen und entsprechen in ihrer Funktion den logarithmierten Wahrscheinlichkeiten eines Naïve-Bayes-Modelles. Sie werden aber nicht aus Häufigkeiten geschätzt, sondern anders gelernt.

Lineare Klassifikatoren

Die Merkmale $f_i(c, d)$ und Gewichte θ_i werden multipliziert und aufsummiert:

$$score(c, d) = \sum_i \theta_i f_i(c, d)$$

Die Werte können direkt zur Klassifikation verwendet werden (linearer Klassifikator)

$$\hat{c} = \arg\max_c score(c, d)$$

Wir wollen aber eine Wahrscheinlichkeitsverteilung definieren (also nicht-negative Werte, die zu 1 summieren)

Log-Lineare Modelle

Die "Scores" sind beliebige reelle Zahlen.

Exponentiation liefert nicht-negative Werte:

$$e^{score(c,d)} = e^{\sum_i \theta_i f_i(c,d)}$$

Normalisierung liefert Werte, die zu 1 summieren:

$$p(c|d) = \frac{1}{Z} e^{\sum_i \theta_i f_i(c,d)} \quad \text{mit } Z = \sum_{c'} e^{\sum_i \theta_i f_i(c',d)}$$

Ein solches Modell heißt log-lineares Modell.

Vektorschreibweise: $p(c|d) = \frac{1}{Z} e^{\theta \cdot \mathbf{f}(c,d)}$

Die Gewichte werden durch Training auf Dokumenten mit bekannter Klasse gelernt.

Log-Lineare Modelle: Training

Ziel des Trainings: Die Modell-Wahrscheinlichkeit der Trainingsdaten soll möglichst groß werden.

- Da die Summe der Wahrscheinlichkeiten konstant ist, müssen die Wahrscheinlichkeiten der nicht beobachteten Daten möglichst klein werden
- Die Wahrscheinlichkeiten können nie 0 werden, da e^x immer positiv ist.

  $\rightarrow$ Parameterglättung nicht nötig

Log-Lineare Modelle: Training

Trainingsdaten: $D = \{(c_1, d_1), (c_2, d_2), ..., (c_N, d_N)\}$

Wir wollen den Gewichtsvektor $\hat{\theta}$ finden, der die bedingte Wahrscheinlichkeit (Likelihood) der korrekten Klassen maximiert:

$$\hat{\theta} = \arg\max_{\theta} \prod_{(c,d) \in D} p_\theta(c|d)$$

Wir schreiben hier $p_\theta(c|d)$, um zu verdeutlichen, dass die Wahrscheinlichkeit von den Parametern θ abhängt. Man kann auch schreiben $p(c|d, \theta)$.

Statt der Wahrscheinlichkeit der Daten maximiert man meist deren Logarithmus (Log-Likelihood):

$$LL_\theta(D) = \sum_{(c,d) \in D} \log p_\theta(c|d)$$

Log-Lineare Modelle: Training

Trainingsdaten: $D = \{(c_1, d_1), (c_2, d_2), ..., (c_N, d_N)\}$

Wir wollen den Gewichtsvektor $\hat{\theta}$ finden, der die bedingte Wahrscheinlichkeit (Likelihood) der korrekten Klassen maximiert:

$$\hat{\theta} = \arg\max_{\theta} \prod_{(c,d) \in D} p_\theta(c|d)$$

Wir schreiben hier $p_\theta(c|d)$, um zu verdeutlichen, dass die Wahrscheinlichkeit von den Parametern θ abhängt. Man kann auch schreiben $p(c|d, \theta)$.

Statt der Wahrscheinlichkeit der Daten maximiert man meist deren Logarithmus (Log-Likelihood):

$$LL_\theta(D) = \sum_{(c,d) \in D} \log p_\theta(c|d)$$

Log-Likelihood

Umformung des Ausdruckes:

$$
\begin{aligned}
LL_\theta(D) &= \sum_{(c,d)\in D} log\ p_\theta(c|d) \\
&= \sum_{(c,d)\in D} log \left(\frac{1}{Z_\theta(d)} e^{\sum_{i=1}^{m} \theta_i f_i(c,d)} \right) \\
&= \sum_{(c,d)\in D} \left(log \frac{1}{Z_\theta(d)} + log\ e^{\sum_{i=1}^{m} \theta_i f_i(c,d)} \right) \\
&= \sum_{(c,d)\in D} \sum_{i=1}^{m} \theta_i f_i(c,d) - \sum_{(c,d)\in D} log\ Z_\theta(d)
\end{aligned}
$$

Um das Maximum der Log-Likelihood in Abhängigkeit von den Parametern θ_j zu finden, setzen wir die partiellen Ableitungen nach θ_j gleich 0.

Log-Likelihood

Umformung des Ausdruckes:

$$
\begin{aligned}
LL_\theta(D) &= \sum_{(c,d)\in D} log\ p_\theta(c|d) \\
&= \sum_{(c,d)\in D} log \left(\frac{1}{Z_\theta(d)} e^{\sum_{i=1}^{m} \theta_i f_i(c,d)} \right) \\
&= \sum_{(c,d)\in D} \left(log \frac{1}{Z_\theta(d)} + log\ e^{\sum_{i=1}^{m} \theta_i f_i(c,d)} \right) \\
&= \sum_{(c,d)\in D} \sum_{i=1}^{m} \theta_i f_i(c,d) - \sum_{(c,d)\in D} log\ Z_\theta(d)
\end{aligned}
$$

Um das Maximum der Log-Likelihood in Abhängigkeit von den Parametern θ_j zu finden, setzen wir die partiellen Ableitungen nach θ_j gleich 0.

Log-Likelihood

Umformung des Ausdruckes:

$$
\begin{aligned}
LL_\theta(D) &= \sum_{(c,d)\in D} log\ p_\theta(c|d) \\
&= \sum_{(c,d)\in D} log\left(\frac{1}{Z_\theta(d)} e^{\sum_{i=1}^{m}\theta_i f_i(c,d)}\right) \\
&= \sum_{(c,d)\in D} \left(log\frac{1}{Z_\theta(d)} + log\ e^{\sum_{i=1}^{m}\theta_i f_i(c,d)}\right) \\
&= \sum_{(c,d)\in D}\sum_{i=1}^{m}\theta_i f_i(c,d) - \sum_{(c,d)\in D} log\ Z_\theta(d)
\end{aligned}
$$

Um das Maximum der Log-Likelihood in Abhängigkeit von den Parametern θ_j zu finden, setzen wir die partiellen Ableitungen nach θ_j gleich 0.

Ableitung der Log-Likelihood nach θ_j

$$\frac{\partial}{\partial \theta_j} \sum_{(c,d) \in D} \sum_{i=1}^{m} \theta_i f_i(c,d) - \sum_{(c,d) \in D} log\ Z_\theta(d) = 0$$

Ableitung des linken Ausdrucks:

$$\frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} \theta_i f_i(c,d) = f_j(c,d)$$

Ableitung des rechten Ausdrucks:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} log\ Z_\theta(d) &= \frac{\partial}{\partial \theta_j} log \sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} \\
&= \frac{1}{\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)}} \left(\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} f_j(c',d) \right) \\
&= \sum_{c'} p_\theta(c'|d)\ f_j(c',d) \\
&= E_{p_\theta(c'|d)}\ f_j(c',d)
\end{aligned}$$

Ableitung der Log-Likelihood nach θ_j

$$\frac{\partial}{\partial \theta_j} \sum_{(c,d) \in D} \sum_{i=1}^{m} \theta_i f_i(c,d) - \sum_{(c,d) \in D} log\ Z_\theta(d) = 0$$

Ableitung des linken Ausdrucks:

$$\frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} \theta_i f_i(c,d) = f_j(c,d)$$

Ableitung des rechten Ausdrucks:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} log\ Z_\theta(d) &= \frac{\partial}{\partial \theta_j} log \sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} \\
&= \frac{1}{\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)}} \left(\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} f_j(c',d) \right) \\
&= \sum_{c'} p_\theta(c'|d)\ f_j(c',d) \\
&= E_{p_\theta(c'|d)}\ f_j(c',d)
\end{aligned}$$

Ableitung der Log-Likelihood nach θ_j

$$\frac{\partial}{\partial \theta_j} \sum_{(c,d) \in D} \sum_{i=1}^{m} \theta_i f_i(c,d) - \sum_{(c,d) \in D} log\ Z_\theta(d) = 0$$

Ableitung des linken Ausdrucks:

$$\frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} \theta_i f_i(c,d) = f_j(c,d)$$

Ableitung des rechten Ausdrucks:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} log\ Z_\theta(d) &= \frac{\partial}{\partial \theta_j} log \sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} \\
&= \frac{1}{\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)}} \left(\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} f_j(c',d) \right) \\
&= \sum_{c'} p_\theta(c'|d)\ f_j(c',d) \\
&= E_{p_\theta(c'|d)}\ f_j(c',d)
\end{aligned}$$

Ableitung der Log-Likelihood nach θ_j

$$\frac{\partial}{\partial \theta_j} \sum_{(c,d) \in D} \sum_{i=1}^{m} \theta_i f_i(c,d) - \sum_{(c,d) \in D} log\ Z_\theta(d) = 0$$

Ableitung des linken Ausdrucks:

$$\frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} \theta_i f_i(c,d) = f_j(c,d)$$

Ableitung des rechten Ausdrucks:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} log\ Z_\theta(d) &= \frac{\partial}{\partial \theta_j} log \sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} \\
&= \frac{1}{\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)}} \left(\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} f_j(c',d) \right) \\
&= \sum_{c'} p_\theta(c'|d)\ f_j(c',d) \\
&= E_{p_\theta(c'|d)}\ f_j(c',d)
\end{aligned}$$

Ableitung der Log-Likelihood nach θ_j

$$\frac{\partial}{\partial \theta_j} \sum_{(c,d) \in D} \sum_{i=1}^{m} \theta_i f_i(c,d) - \sum_{(c,d) \in D} log\ Z_\theta(d) = 0$$

Ableitung des linken Ausdrucks:

$$\frac{\partial}{\partial \theta_j} \sum_{i=1}^{m} \theta_i f_i(c,d) = f_j(c,d)$$

Ableitung des rechten Ausdrucks:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} log\ Z_\theta(d) &= \frac{\partial}{\partial \theta_j} log \sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} \\
&= \frac{1}{\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)}} \left(\sum_{c'} e^{\sum_{i=1}^{m} \theta_i f_i(c',d)} f_j(c',d) \right) \\
&= \sum_{c'} p_\theta(c'|d)\ f_j(c',d) \\
&= E_{p_\theta(c'|d)}\ f_j(c',d)
\end{aligned}$$

Log-Lineare Modelle

Zusammensetzen der Teilergebnisse:

$$\frac{\partial LL_\theta(D)}{\partial \theta_j} = \sum_{(c,d)\in D} f_j(c,d) - \sum_{(c,d)\in D} E_{p_\theta(c'|d)} f_j(c',d) = 0$$

$$\sum_{(c,d)\in D} f_j(c,d) = \sum_{(c,d)\in D} E_{p_\theta(c'|d)} f_j(c',d)$$

$\Rightarrow$ Die erwartete Häufigkeit jedes Merkmales muss gleich der beobachteten Häufigkeit werden.

$\Rightarrow$ Die partielle Ableitung ist gleich der Differenz der beiden Häufigkeiten.

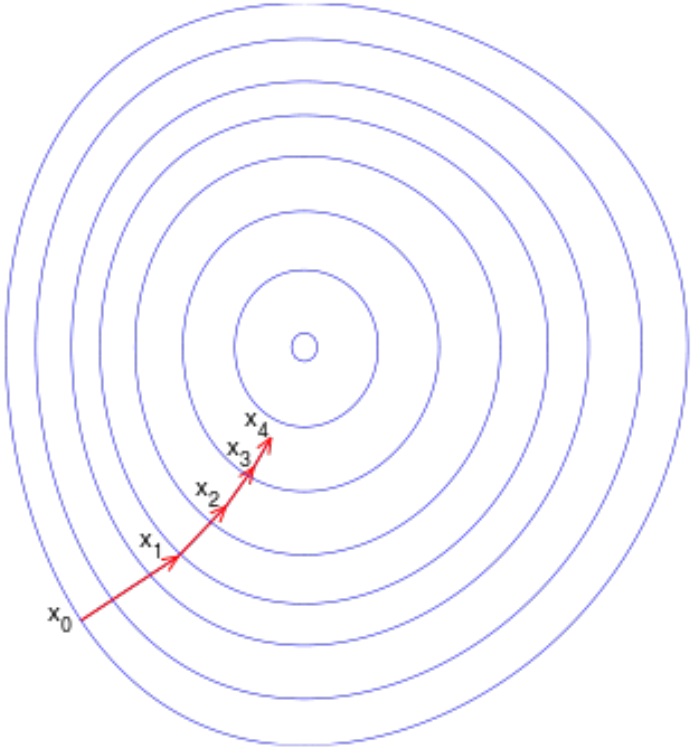
$\Rightarrow$ Der Gradient $\nabla LL_\theta(D)$ fasst alle partiellen Ableitungen zu einem Vektor zusammen.

Training mit Gradientenanstieg

Um die Likelihood zu erhöhen, müssen wir die Parameter in Richtung steigender Likelihood verändern (Gradientenanstieg).

Algorithmus

initialize θ
for N iterations do
$\theta \leftarrow \theta + \eta \nabla LL_\theta(D)$



aus Wikipedia

Die Lernrate η bestimmt, wie groß die einzelnen Schritte sind.

Wenn eine Funktion minimiert werden muss, spricht man von Gradientenabstieg.

Varianten von Gradientenanstieg

- (Batch-)Gradientenanstieg
  - berechnet die Ableitung für alle Daten zusammen
  - Anpassung der Gewichte nachdem alle Daten verarbeitet wurden
  - konvergiert langsam
- Stochastischer Gradientenanstieg
  - berechnet die Ableitung einzeln für jeden Datensatz
  - passt die Gewichte nach jedem Datensatz an
  - lernt schneller, aber konvergiert nicht
- Minibatch-Gradientenanstieg
  - berechnet die Ableitung für eine bestimmte Zahl von Datensätzen
  - passt die Gewichte nach der Verarbeitung jedes solchen Minibatches an
  - Eigenschaften zwischen Batch GD und SGD je nach Minibatchgröße

Overfitting

Bei begrenzten Trainingsdaten besteht die Gefahr des Overfittings.

Beipiel: (fälschlich als negativ klassifizierte Filmbewertung in Trainingsdaten)

Dieser Film ist unglaublich spannend. Die Hauptfigur Hugh Glass wird von Leonardo DiCaprio gespielt.

Wenn das Wort *Glass* nur in dieser Rezension auftaucht, dann wird das Gewicht des Merkmals *(Glass, negativ)* im Training so lange erhöht, bis die Rezension als negativ klassifiziert wird.

Das Problem taucht nicht nur bei Annotationsfehlern in den Trainingsdaten auf, sondern auch dann, wenn die vorhandenen Merkmale nicht alle notwendigen Informationen repräsentieren.

Regularisierung

Gegenmaßnahme:

Ziel-Funktion so modifizieren, dass große Gewichte ,,bestraft" werden:

$$LL_\theta(D) - \frac{\mu}{2}\sum_i \theta_i^2 \quad (L_2\text{-Regularisierung})$$

$$LL_\theta(D) - \mu\sum_i |\theta_i| \quad (L_1\text{-Regularisierung})$$

$\Rightarrow$ Große Gewichte sind nur erlaubt, wenn sie eine große Verbesserung der Likelihood bewirken.

Regularisierung

neuer Trainingsalgorithmus

initialize θ
for N iterations do

$$\delta \leftarrow \begin{cases} \mu\theta & \text{falls } L_2\text{-Regularisierung} \\ \mu sign(\theta) & \text{falls } L_1\text{-Regularisierung} \end{cases}$$

$$\theta \leftarrow \theta + \eta(\nabla LL_\theta(D) - \delta)$$

weitere Regularisierungsmethode (kombinierbar)

- Zahl der Trainingsiterationen mit Heldout-Daten optimieren

Andere Lernalgorithmen

Neben Gradientenanstieg gibt es weitere Trainingsverfahren

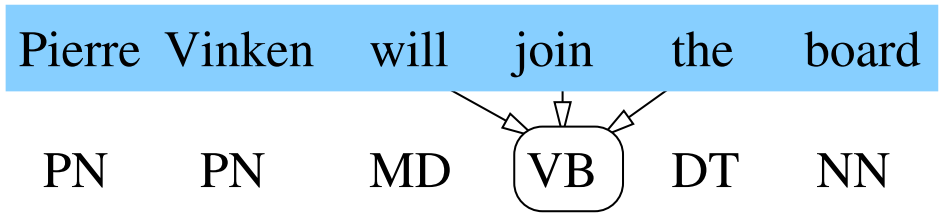
- generalized iterative scaling (nicht mehr oft genutzt)
- conjugate gradient
- L-BFGS (limited memory Broyden-Fletcher–Goldfarb-Shannon)
  (Batchverfahren, schnell konvergierend, erfordert viel Speicherplatz)

Taggen mit log-linearen Modellen

Log-lineare Modelle können auch zum Wortart-Taggen verwendet werden.

Es gibt 2 Möglichkeiten, einen Wort-Tagger zu implementieren:

- Jedes Tag wird unabhängig von allen anderen Tags zugewiesen

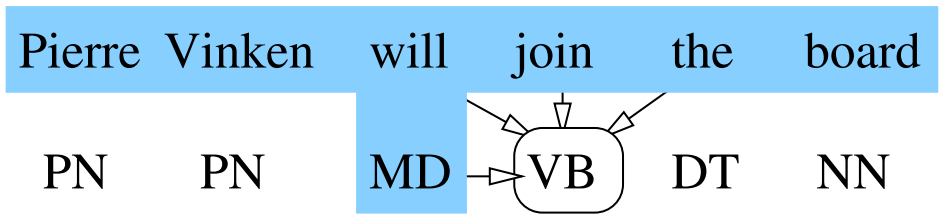


  Problem: inkonsistente Ausgaben

  Beispiel: Das ist die/Nom beste/Nom Lösung/Akk ./$.

- Alle Wörter werden gemeinsam desambiguiert (analog HMMs)

  Dabei hängt jedes Tag vom vorherigen Tag ab.



  → (Linear Chain) Conditional Random Field

Conditional Random Field

Für jede Wortposition wird eine Menge von Merkmalen extrahiert

1. lexikalische Merkmale (ohne Nachbartag-Information)
   Wort+Tag, Wortsuffix+Tag, Wort"shape"+Tag, voriges Wort+Tag, nächstes Wort+Wort+Tag, Wort+Tag im Lexikon
2. Kontext-Merkmale (mit Nachbartag-Information)
   voriges Tag+Tag, letzte2Tags+Tag, voriges Tag+Wort+Tag

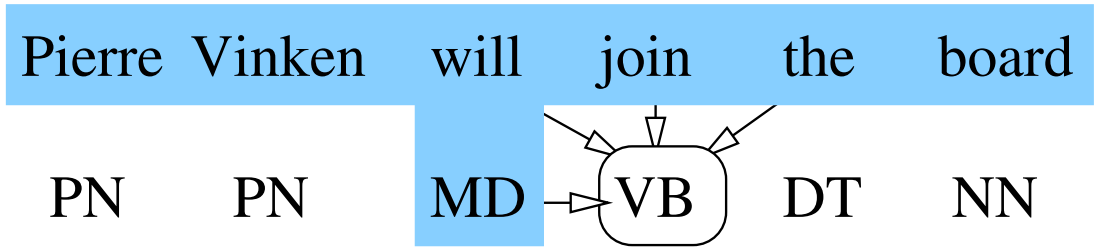
Die Merkmale f_k werden mit Gewichten θ_k multipliziert und über alle Positionen aufsummiert.

$$p(t_1^n|w_1^n) = \frac{1}{Z(w_1^n)} e^{\sum_{i=1}^{n+1} \sum_k \theta_k f_k(t_{i-m}^i, w_1^n, i)}$$

m: Zahl der Vorgängertags, von denen das nächste Tag abhängt

i iteriert bis $n+1$, weil ein Satzende-Tag $\langle/s\rangle$ hinzugefügt wird.

Conditional Random Fields



Jedes Tag darf abhängen von

- beliebigen Wörtern (einfaches log-lineares Modell)
- zusätzlich vom letzten Tag (LC-CRF 1. Ordnung)
- zusätzlich vom vorletzten Tag (LC-CRF 2. Ordnung)

wichtig: Bei einem LC-CRF sind Algorithmen der dynamischen Programmierung anwendbar (Viterbi, Forward-Backward)

Conditional Random Fields

Bezug zu Hidden-Markow-Modellen

$$\hat{t_1^n} = \arg\max_{t_1^n} p(t_1^n|w_1^n) = \arg\max_{t_1^n} \frac{1}{Z(w_1^n)} e^{\sum_{i=1}^{n+1}\sum_k \theta_k f_k(t_{i-m}^i, w_1^n, i)}$$

$$= \arg\max_{t_1^n} e^{\sum_{i=1}^{n+1}\sum_k \theta_k f_k(t_{i-m}^i, w_1^n, i)} \quad \text{Normalisierungskonstante ignorieren}$$

$$= \arg\max_{t_1^n} \prod_{i=1}^{n+1} e^{\sum_k \theta_k f_k(t_{i-m}^i, w_1^n, i)}$$

$$= \arg\max_{t_1^n} \prod_{i=1}^{n+1} \underbrace{e^{\sum_k \theta_k^C f_k^C(t_{i-m}^i, w_1^n, i)}}_{\text{Kontextmerkmale}} \underbrace{e^{\sum_k \theta_k^L f_k^L(t_i, w_1^n, i)}}_{\text{lexikalische Merkmale}}$$

- Die Formel ähnelt der HMM-Formel $\arg\max_{t_1^n} \prod_{i=1}^{n+1} p(t_i|t_{i-m})\ p(w_i|t_i)$
- Annotiert wird mit dem Viterbi-Algorithmus (analog HMMs).
- $\prod_i e^{\sum_k \cdots}$ kann durch $\sum_i \sum_k \ldots$ ersetzt werden (log. Darstellung)

Viterbi-Algorithmus (Bigramm-Tagger)

berechnet die beste Tagfolge (hier mit logarithmierten Viterbiwahrsch.)

lokaler Score: $s(t', t, w_1^n, i) := \sum_k \theta_k f_k(t', t, w_1^n, i)$

1. Initialisierung: $\delta_t(0) = \begin{cases} 0 & \text{falls } t = \langle s \rangle \\ -inf & \text{sonst} \end{cases}$

2. Berechnung:
$$\delta_t(i) = \max_{t'} \delta_{t'}(i-1) + s(t', t, w_1^n, i)$$
$$\psi_t(i) = \arg\max_{t'} \delta_{t'}(i-1) + s(t', t, w_1^n, i)$$

3. Ausgabe
$$t_{n+1} = \langle /s \rangle$$
$$t_{i-1} = \psi_{t_i}(i)$$

t und t' sind Tags, i ist eine Wortposition, $\delta_t(i)$ ein Viterbiwert und $\psi_t(i)$ das beste Vorgängertag.

Training von Conditional Random Fields

- Für das Training mit Gradientenanstieg werden die erwarteten Merkmalshäufigkeiten benötigt.
  - Wie bei den HMMs werden diese mit dem Forward-Backward-Algorithmus berechnet.
  - Hier wird er aber für überwachtes Training eingesetzt.
- Die erwarteten Häufigkeiten werden von den beobachteten Häufigkeiten in den Trainingsdaten subtrahiert, um den Gradienten zu erhalten.
- Der Gradient wird mit der Lernrate multipliziert und zum Gewichtsvektor addiert. (Gradientenanstieg)
- Diese Schritte werden N-mal wiederholt.

Training von Conditional Random Fields

$$p(t_1^n|w_1^n) = \frac{1}{Z(w_1^n)} \prod_{i=1}^{n+1} e^{\sum_k \theta_k f_k(t_{i-m}^i, w_1^n, i)}$$

Zur Berechnung der erwarteten Merkmalshäufigkeiten eines lex. Merkmals muss berechnet werden:

$$\frac{\sum \text{ Wk. aller Tagfolgen mit Tag } t \text{ an Position } i}{\sum \text{Wk. aller Tagfolgen insgesamt}}$$

- Die Normalisierungskonstanten $Z(w_1^n)$ können im Zähler und Nenner aus der Summe ausgeklammert und gekürzt werden.
- Die Summe im Zähler wird als Produkt von Forward- und Backward-Wahrscheinlichkeiten berechnet.
- Die Summe im Nenner (ohne Normalisierungskonstante) ist gleich dem Wert der Forward-Funktion für das Satzende-Tag.

Forward-Backward-Algorithmus (Modell 1. Ordnung)

Wenn T die Menge der Tags ist und $w_1^n = w_1 ... w_n$ die Wortfolge des Eingabesatzes der Länge n und θ_k das Gewicht des Merkmales k und $f_k(...)$ der Wert des Merkmales k, dann werden die Forward-Wahrscheinlichkeiten $\alpha_t(i)$, die Backward-Wahrscheinlichkeiten $\beta_t(i)$ und die Aposteriori-Wahrscheinlichkeiten $\gamma_t(i)$ des Tags $t \in T$ an der Satzposition i wie folgt berechnet:

$$z(t', t, w_1^n, i) := e^{s(t', t, w_1^n, i)} = e^{\sum_k \theta_k f_k(t', t, w_1^n, i)}$$

$$\begin{aligned}
\alpha_t(0) &= 1 \text{ falls } t = \langle s \rangle \text{ sonst } 0 \\
\alpha_t(i) &= \sum_{t' \in T} \alpha_{t'}(i-1)\, z(t', t, w_1^n, i) \quad \text{für alle } t \in T \text{ und } 0 < i \leq n+1 \\
\beta_t(n+1) &= 1 \text{ falls } t = \langle /s \rangle \text{ sonst } 0 \\
\beta_t(i-1) &= \sum_{t' \in T} \beta_{t'}(i)\, z(t, t', w_1^n, i) \quad \text{für alle } t \in T \text{ und } 0 < i \leq n+1 \\
\gamma_t(i) &= \frac{\alpha_t(i)\, \beta_t(i)}{\alpha_{\langle /s \rangle}(n+1)} \quad \text{für alle } t \in T \text{ und } 0 < i \leq n+1 \\
\gamma_{tt'}(i) &= \frac{\alpha_t(i-1)\, z(t, t', w_1^n, i)\, \beta_{t'}(i)}{\alpha_{\langle /s \rangle}(n+1)} \text{ für alle } t, t' \in T \text{ und } 0 < i \leq n+1
\end{aligned}$$

LC-CRFs

Hauptaufgabe bei der Entwicklung von LC-CRFs

- gute Merkmale finden

  $\rightarrow$ "Feature Engineering", linguistische Intuition hilfreich

- optimale Kombination der Merkmale bestimmen

  weniger wichtig bei gut funktionierender Regularisierung

Evaluierung

Die Evaluierung eines Wortart-Taggers erfordert ein manuell annotiertes **Testkorpus**, das nicht Teil der Trainingsdaten war.

1. Training des Taggers auf den Trainingsdaten
2. Taggen der Wortfolge des Testkorpus mit dem Tagger
3. Vergleich der Taggerausgabe mit den Tags im Korpus
4. Genauigkeit = #korrekte Tags / #Wörter

Signifikanztest

Beim Vergleich eines verbesserten Taggers mit einem existierenden ist es wichtig zu wissen, ob die Verbesserung der Genauigkeit **signifikant** ist.

$\Rightarrow$ Statistischer Test

Angenommen

- Bei 100 Wörtern lag genau einer der Tagger richtig
- Bei 60 dieser Wörter lag der neue Tagger richtig
- Bei 40 dieser Wörter lag der alte Tagger richtig

Nullhypothese: Der neue Tagger ist nicht besser als der alte Tagger

Dies ist ein einseitiger Binomialtest. Wir berechnen:

$$b(\geq 60, 100, 0.5) \approx 0.03$$

$\Rightarrow$ Der Unterschied ist signifikant (< 0.05).

Optimierung der Metaparameter

Der CRF-Tagger hat eine Reihe von **Metaparametern** (Liste der Merkmale, Lernrate, Regularisierung, Zahl der Trainingsiterationen), die beliebig gewählt werden können.

Um gute Werte auszuwählen, kann die Genauigkeit des Taggers für die einzelnen Parameterwerte auf Testdaten evaluiert werden, die aber nicht mit den Testdaten für die eigentliche Evaluierung identisch sein dürfen.

Solche Daten für die Parameter-Optimierung nennt man **Entwicklungsdaten** (Developmentdaten).

In der Regel braucht man für Experimente in der Computerlinguistik **Trainingsdaten**, **Developmentdaten** und **Testdaten**.

Optimierung von Code

2 Regeln

1. Don't optimize!
2. If you have to do it, do it later!

Meine Zusatzregel

1. Wenn zwei Lösungen ansonsten vergleichbar sind, wählen Sie die effizientere!

Vorgehen bei der Code-Optimierung

1. Entwickeln Sie Ihren Code ohne besondere Optimierungsmaßnahmen.
2. Testen Sie, ob der Code effizient genug ist.
3. Falls nicht
   - Analysieren Sie Ihren Code mit einem Profiler.
   - Ermitteln Sie die Funktion, welche die meiste Rechenzeit verbraucht.
   - Optimieren Sie diese Funktion.
   - Weiter mit 2)

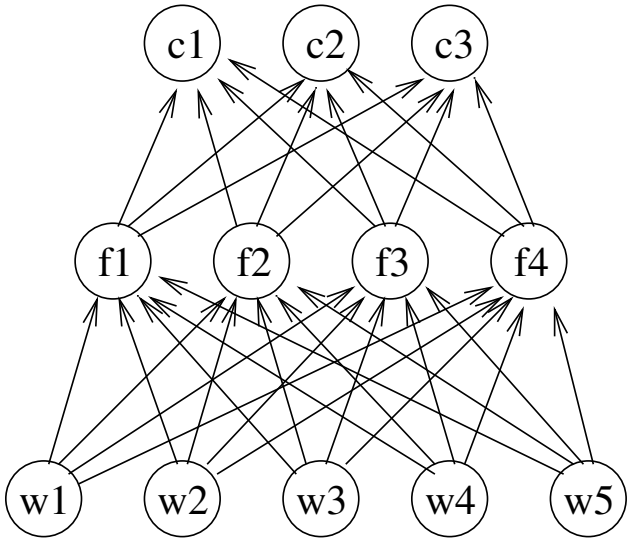
Lernen von Repräsentationen

- Eine gute Menge von Merkmalen zu finden ist sehr zeitaufwendig.
- Die gefundene Menge ist selten optimal.
- Die Arbeit muss neu gemacht werden, wenn sich die Aufgabe ändert.

Könnte man die Merkmale automatisch lernen?

Von log-linearen Modellen zu neuronalen Netzen

grafische Darstellung eines log-linearen Modelles



Klassen: c1, c2, c3
Merkmale: f1, f2, f3, f4
Eingabewörter: w1, w2, w3, w4, w5

Jedes Merkmal f_k wird aus w_1^5 berechnet

Jede Klasse c_k wird aus f_1^4 berechnet mit

$$c_k = \frac{1}{Z} e^{\sum_i \theta_{ki} f_i} = \mathit{softmax}\left(\sum_i \theta_{ki} f_i\right)$$

Anmerkung: Hier sind die Merkmale unabhängig von den Klassen. Stattdessen gibt es für jedes Merkmal klassenspezifische Gewichte.

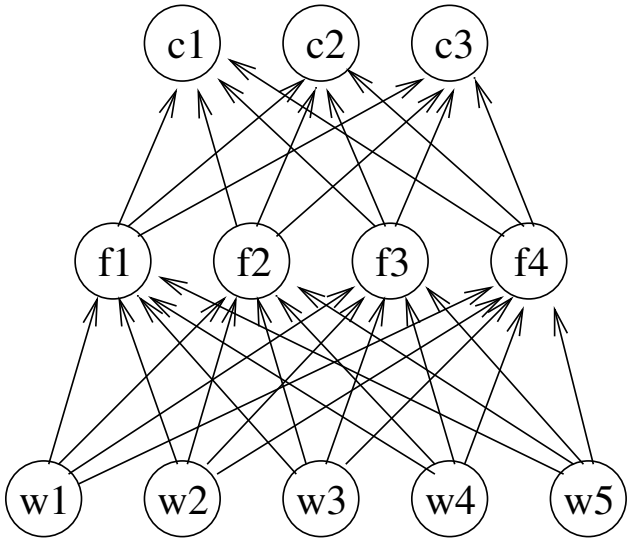
Idee: Die Merkmale f_i werden analog zu den Klassen c_i berechnet/gelernt:

$$f_k = act\left(\sum_i \theta_{ki} w_i\right) = act(net_k)$$

$\Rightarrow$ neuronales Netz

Von log-linearen Modellen zu neuronalen Netzen

grafische Darstellung eines log-linearen Modelles



Klassen: c1, c2, c3
Merkmale: f1, f2, f3, f4
Eingabewörter: w1, w2, w3, w4, w5

Jedes Merkmal f_k wird aus w_1^5 berechnet

Jede Klasse c_k wird aus f_1^4 berechnet mit

$$c_k = \frac{1}{Z} e^{\sum_i \theta_{ki} f_i} = softmax\left(\sum_i \theta_{ki} f_i\right)$$

Anmerkung: Hier sind die Merkmale unabhängig von den Klassen. Stattdessen gibt es für jedes Merkmal klassenspezifische Gewichte.

Idee: Die Merkmale f_i werden analog zu den Klassen c_i berechnet/gelernt:

$$f_k = act\left(\sum_i \theta_{ki} w_i\right) = act(net_k)$$

$\Rightarrow$ neuronales Netz

Embeddings

$$f_k = act\left(\sum_i \theta_{ki} w_i\right)$$

Die Eingabe w_i muss eine numerische Repräsentation des Wortes sein.

Möglichkeiten

1. binärer Vektor (1-hot-Repräsentation)
   - Beispiel: $w_i^{hot} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$
   - Dimension = Vokabulargröße
     $\Rightarrow$ Alle Wörter sind sich gleich ähnlich/unähnlich
2. reeller Vektor (verteilte Repräsentation, distributed representation)
   - Beispiel: $w_i^{dist} = (3.275, -7.295, 0.174, 5.7332)$
   - Vektorlänge beliebig wählbar
   - Ziel: Ähnliche Wörter durch ähnliche Vektoren repräsentieren

Embeddings

$$f_k = act\left(\sum_i \theta_{ki} w_i\right)$$

Die Eingabe w_i muss eine numerische Repräsentation des Wortes sein.

Möglichkeiten

1. binärer Vektor (1-hot-Repräsentation)
   - Beispiel: $w_i^{hot} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$
   - Dimension = Vokabulargröße
     $\Rightarrow$ Alle Wörter sind sich gleich ähnlich/unähnlich
2. reeller Vektor (verteilte Repräsentation, distributed representation)
   - Beispiel: $w_i^{dist} = (3.275, -7.295, 0.174, 5.7332)$
   - Vektorlänge beliebig wählbar
   - Ziel: Ähnliche Wörter durch ähnliche Vektoren repräsentieren

Embeddings

$$f_k = act\left(\sum_i \theta_{ki} w_i\right)$$

Die Eingabe w_i muss eine numerische Repräsentation des Wortes sein.

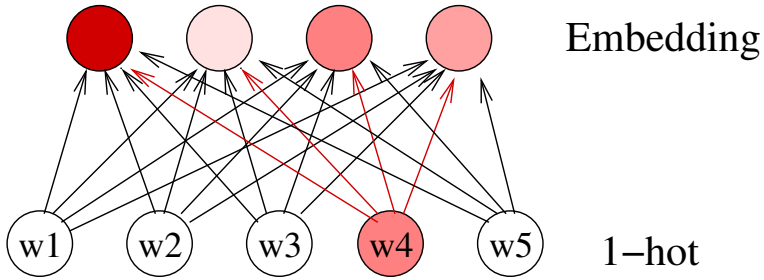
Möglichkeiten

1. binärer Vektor (1-hot-Repräsentation)
   - Beispiel: $w_i^{hot} = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0)$
   - Dimension = Vokabulargröße
     $\Rightarrow$ Alle Wörter sind sich gleich ähnlich/unähnlich

2. reeller Vektor (verteilte Repräsentation, distributed representation)
   - Beispiel: $w_i^{dist} = (3.275, -7.295, 0.174, 5.7332)$
   - Vektorlänge beliebig wählbar
   - Ziel: Ähnliche Wörter durch ähnliche Vektoren repräsentieren

Embeddings

Wir können die 1-hot-Repräsentation in einem neuronalen Netz auf die verteilte Repräsentation (Embedding) abbilden:



Da nur ein 1-hot-Neuron den Wert 1 und alle anderen den Wert 0 haben, ist die verteilte Repräsentation identisch zu den Gewichten des aktiven Neurons. (Die Aktivierungsfunktion ist hier die Identität $act(x) = x$)

Die Embeddings werden dann ebenfalls im Training gelernt.

Lookup-Tabelle

Wir können die verteilten Repräsentationen der Wörter zu einer Matrix L (Embedding-Matrix, Lookup-Tabelle) zusammenfassen. Multiplikation eines 1-hot-Vektors mit dieser Matrix liefert die verteilte Repräsentation.

$$\boldsymbol{w}^{dist} = L\boldsymbol{w}^{hot}$$

Matrix-Vektor-Multiplikation:

$$\begin{array}{cccccc|c} & & & & & & 0 & \\ & & & & & & 0 & \\ & & & & & & 1 & \boldsymbol{w}^{hot} \\ & & & & & & 0 & \\ & & & & & & 0 & \\ \hline & 3.3 & 8.5 & 7.3 & 1.9 & -7.7 & 7.3 & \\ & 4.8 & 5.1 & 4.7 & -5.3 & 3.1 & 4.7 & \\ L & -8.6 & 8.7 & 9.5 & 3.5 & 5.6 & 9.5 & \boldsymbol{w}^{dist} \\ & 6.9 & 6.4 & -4.3 & 5.7 & 3.3 & -4.3 & \\ & 7.7 & 6.7 & -8.2 & 9.7 & -9.1 & -8.2 & \end{array}$$

Aktivierungsfunktionen

Neuron: $$f_k = act\left(\sum_i \theta_{ki} w_i\right) = act(net_k)$$

In der Ausgabeebene eines neuronalen Netzes wird (wie bei log-linearen Modellen) oft die Softmax-Aktivierungsfunktion verwendet.

$$c_k = softmax(net_k) = \frac{1}{Z} e^{net_k} \quad \text{mit } Z = \sum_{k'} e^{net_{k'}}$$

Spezialfall: 2 Klassen

SoftMax: $c_k = \frac{1}{Z} e^{net_k}$ mit $net_k = \sum_i \theta_{ki} f_i = \boldsymbol{\theta}_k^T \boldsymbol{f} = \boldsymbol{\theta}_k \cdot \boldsymbol{f}$

Im Falle von zwei Klassen c_1 und c_2 ergibt sich:

$$\begin{aligned} c_1 &= \frac{e^{\boldsymbol{\theta}_1 \cdot \boldsymbol{f}}}{e^{\boldsymbol{\theta}_1 \cdot \boldsymbol{f}} + e^{\boldsymbol{\theta}_2 \cdot \boldsymbol{f}}} \frac{e^{-\boldsymbol{\theta}_1 \cdot \boldsymbol{f}}}{e^{-\boldsymbol{\theta}_1 \cdot \boldsymbol{f}}} \\ &= \frac{1}{1 + e^{-(\boldsymbol{\theta}_1 - \boldsymbol{\theta}_2) \cdot \boldsymbol{f}}} \end{aligned}$$

Nach einer Umparametrisierung erhalten wir die Sigmoid-Funktion

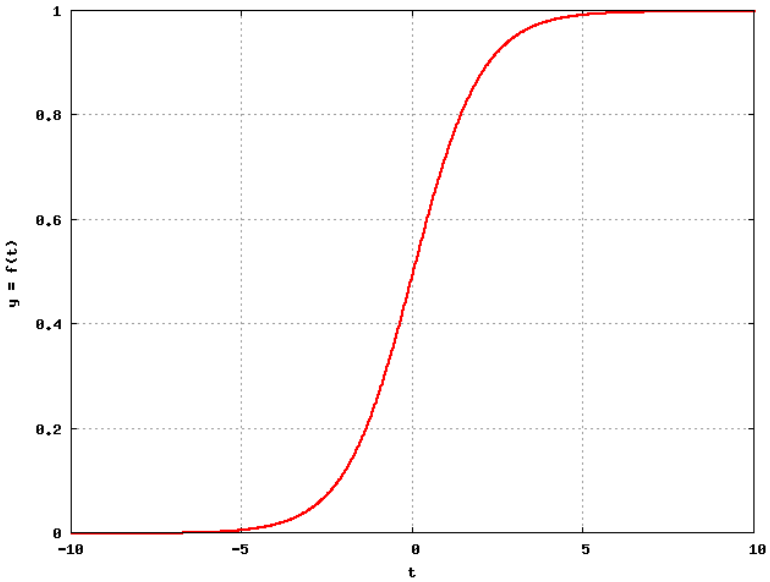
$$\begin{aligned} c_1 &= \frac{1}{1 + e^{-\theta f}} \quad \text{mit } \theta = \theta_1 - \theta_2 \\ c_2 &= 1 - c_1 \end{aligned}$$

$\Rightarrow$ Bei 2-Klassen-Problemen genügt ein Neuron.

Sigmoid-Funktion

Die Sigmoid-Funktion wird häufig als Aktivierungsfunktion der Neuronen in den versteckten Ebenen (hidden layers) genommen.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



Man könnte also sagen, dass die versteckten Neuronen die Wahrscheinlichkeiten von binären Merkmalen berechnen.

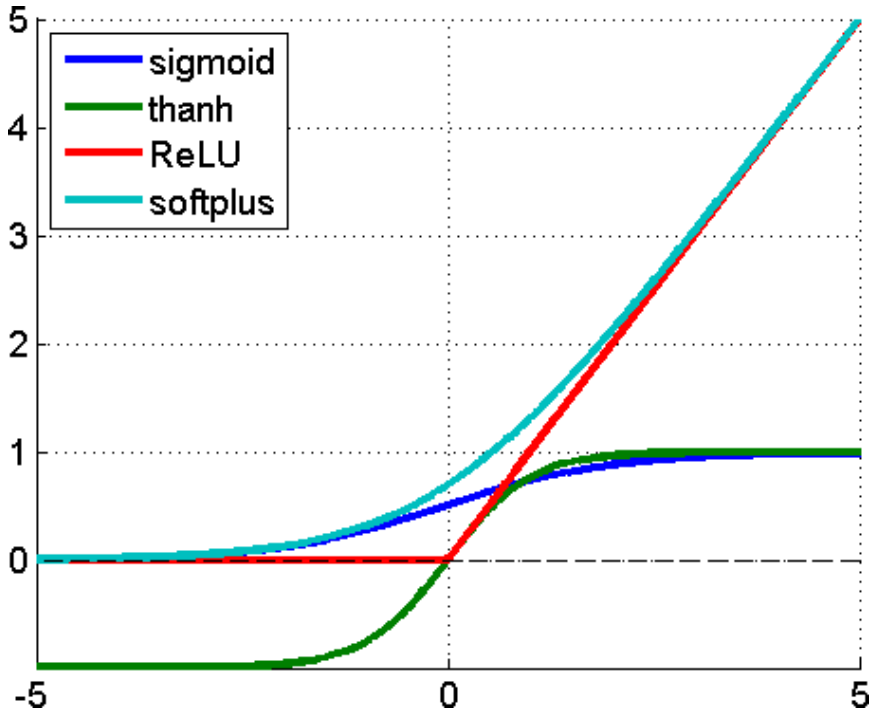
Verschiedene Aktivierungsfunktionen

sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$

tanh: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$

ReLU: $ReLU(x) = max(0, x)$

Softplus: $softplus(x) = ln(1 + e^x)$



Der Tangens Hyperbolicus tanh: skalierte und verschobene Sigmoidfunktion, oft besser als Sigmoidfunktion, liefert 0 als Ausgabe, wenn alle Eingaben 0 sind

ReLUs (rectified linear units): einfach zu berechnen, bei 0 nicht differenzierbar.

Softplus ist eine überall differenzierbare Annäherung an ReLU.

Ableitungen der Aktivierungsfunktionen

sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x}$

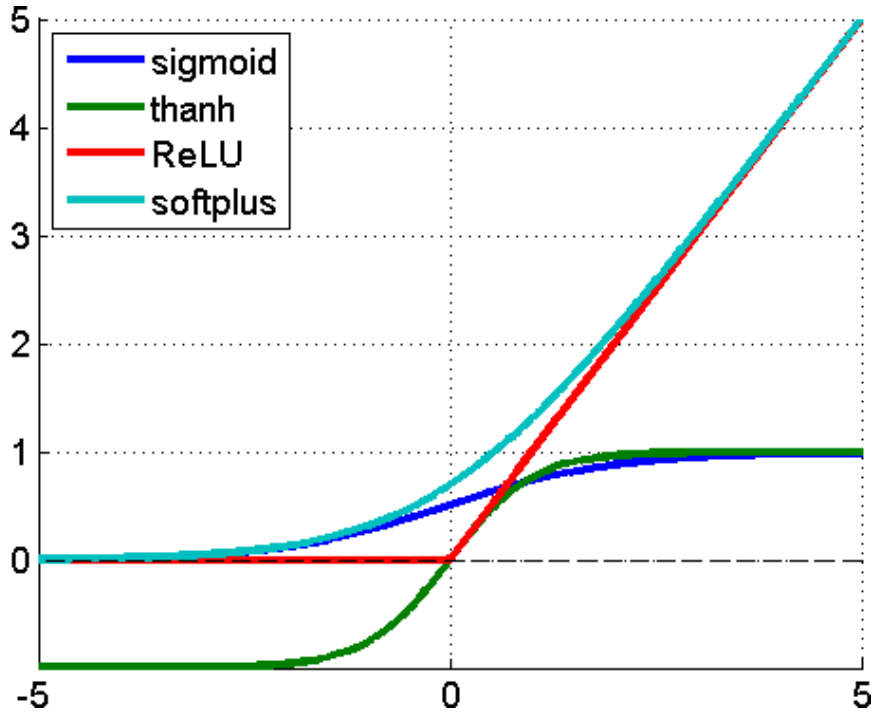
$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

tanh: $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\sigma(2x) - 1$

$$tanh'(x) = 1 - \tanh(x)^2$$

ReLU: $ReLU(x) = max(0, x)$

$$ReLU'(x) = \begin{cases} 1 & \text{falls } x > 0 \\ 0 & \text{falls } x < 0 \end{cases}$$



Warum ist die Aktivierungsfunktion wichtig?

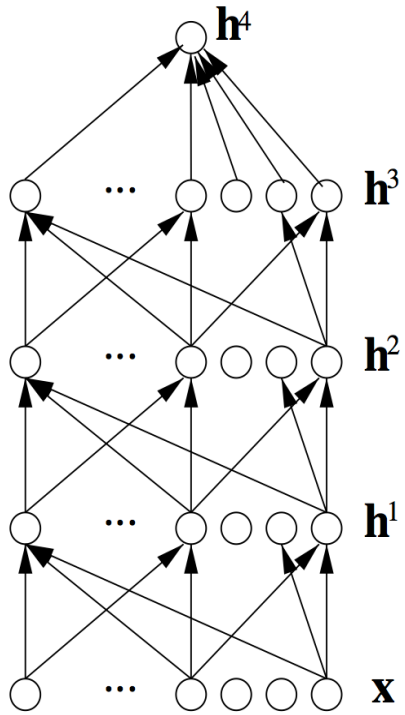
Ohne (bzw. mit linearer) Aktivierungsfunktion gilt:

- Jede Ebene eines neuronalen Netzes führt eine lineare Transformation der Eingabe durch (= Multiplikation mit einer Matrix).
- Eine Folge von linearen Transformationen kann immer durch eine einzige lineare Transformation ersetzt werden.

$\Rightarrow$ Jedes mehrstufige neuronale Netz mit linearer Aktivierungsfunktion kann durch ein äquivalentes einstufiges Netz ersetzt werden.

$\Rightarrow$ Mehrstufige Netze machen also nur Sinn, wenn nicht-lineare Aktivierungsfunktionen verwendet werden.
Ausnahme: Embedding-Schicht mit gekoppelten Parametern

Neuronale Netze



Die Aktivierung a_k eines Neurons ist gleich der Summe der gewichteten Eingaben e_i moduliert durch eine Aktivierungsfunktion *act*.

$$a_k = act(\sum_i w_{ik} e_i + b_k)$$

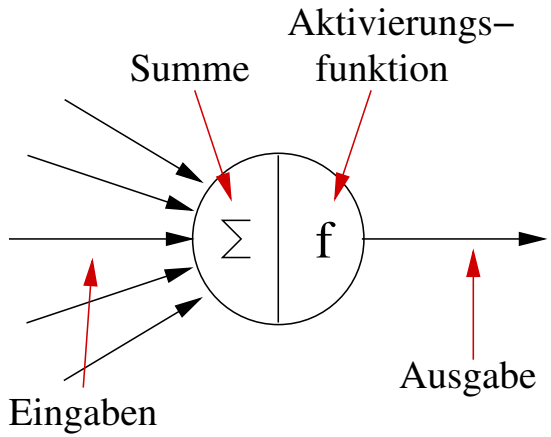
Der zusätzliche Biasterm b_k ermöglicht, dass bei einem Nullvektor als Eingabe die Ausgabe nicht-null ist.

Vektorschreibweise $a_k = act\left(\boldsymbol{w_k}^T \boldsymbol{e} + b_k\right)$
Matrixschreibweise $\boldsymbol{a} = act\left(W\boldsymbol{e} + \boldsymbol{b}\right)$

wobei *act* elementweise angewendet wird:

$$act([z_1, z_2, z_3]) = [act(z_1), act(z_2), act(z_3)]$$

trainierbare Parameter: $W, \boldsymbol{b}$



Zielfunktionen

Die verwendeten Ausgabefunktionen und Zielfunktionen hängen von der Aufgabe ab.

- **Regression:** Ein Ausgabeneuron, welches einen Zahl liefert
  Beispiel: Vorhersage der Restlebenszeit eines Patienten in Abhängigkeit von verschiedenen Faktoren wie Alter, Blutdruck, Rauch- und Essgewohnheiten, Sportaktivitäten etc.
  Aktivierungsfunktion der Ausgabeneuronen: $act(x) = x$ (Identität)
  Kostenfunktion: $(y - o)^2$ quadrierter Abstand zwischen gewünschter Ausgabe y und tatsächlicher Ausgabe o
- **Klassifikation:** n mögliche disjunkte Ausgabeklassen repräsentiert durch je ein Ausgabeneuron
  Beispiel: Diagnose der Krankheit eines Patienten in Abhängigkeit von Symptomen
  Aktivierungsfunktion der Ausgabeneuronen: softmax
  Kostenfunktion: $log(y^T o)$ (Loglikelihood, Crossentropie)

Zielfunktionen

Die verwendeten Ausgabefunktionen und Zielfunktionen hängen von der Aufgabe ab.

- **Regression:** Ein Ausgabeneuron, welches einen Zahl liefert
  Beispiel: Vorhersage der Restlebenszeit eines Patienten in Abhängigkeit von verschiedenen Faktoren wie Alter, Blutdruck, Rauch- und Essgewohnheiten, Sportaktivitäten etc.
  Aktivierungsfunktion der Ausgabeneuronen: $act(x) = x$ (Identität)
  Kostenfunktion: $(y - o)^2$ quadrierter Abstand zwischen gewünschter Ausgabe y und tatsächlicher Ausgabe o
- **Klassifikation:** n mögliche disjunkte Ausgabeklassen repräsentiert durch je ein Ausgabeneuron
  Beispiel: Diagnose der Krankheit eines Patienten in Abhängigkeit von Symptomen
  Aktivierungsfunktion der Ausgabeneuronen: softmax
  Kostenfunktion: $log(y^T o)$ (Loglikelihood, Crossentropie)

Deep Learning

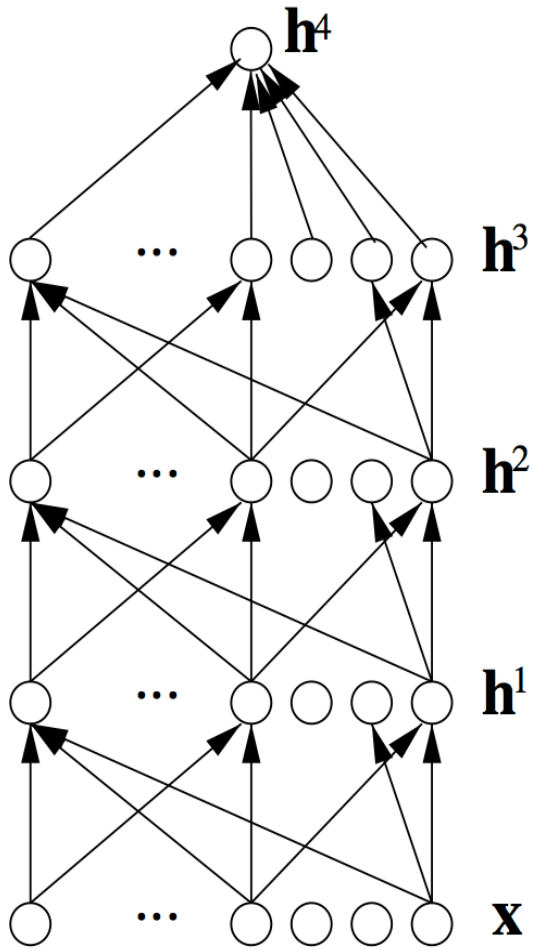
Neuronale Netze mit mehreren versteckten Ebenen nennt man tiefe Netze → Deep Learning.

Bei der Anwendung eines neuronalen Netzes werden die Eingabewerte **x** über die Neuronen der versteckten Ebenen $h_1, ..., h_3$ bis zur Ausgabeebene h_4 propagiert.

Je höher die versteckte Ebene, desto komplexer sind die darin repräsentierten Merkmale.

Im Training werden alle Parameter (Gewichte, Biases) gleichzeitig optimiert. Dazu wird der Gradient der Zielfunktion an den Ausgabeneuronen berechnet und bis zu den Eingabeneuronen zurückpropagiert.

→ Backpropagation



Neuronale Netze (Deep Learning)

Neuronale Netze (NN) sind keine neue Erfindung.
Warum werden sie erst jetzt in großem Stil in der Computerlinguistik verwendet?

- Schnellere Hardware erlaubt das Training großer NN
- NN passen ideal zu aktuellen Hardware-Trends (Parallelisierung)

  Die Grundoperation eines neuronalen Netzes ist die Multiplikation zweier großer Matrizen → ideale Aufgabe für GPUs (Grafikkarten)
- NN profitieren von den großen Datenmengen, die verfügbar werden
- Neue Netzarchitekturen (LSTMs, Highway Networks, Residual Networks) und Initialisierungsmethoden (Glorot-Initialisierung) ermöglichen das Training von tiefen und rekurrenten Netzen.

Neuronale Netze sind ideal für einige scheinbar einfache Aufgaben, die Maschinen Probleme bereiten:

gehen, Objekte erkennen, gesprochene Sprache verstehen, radfahren usw.

Neuronale Netze in der Spracherkennung

Dahl et al. (2010): Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition

Akustikmodell	RT03S FSH	Hub5 SWB
traditionelle Merkmale	27,4	23,6
Deep Learning	18,5	16,1
Verbesserung	33%	32%

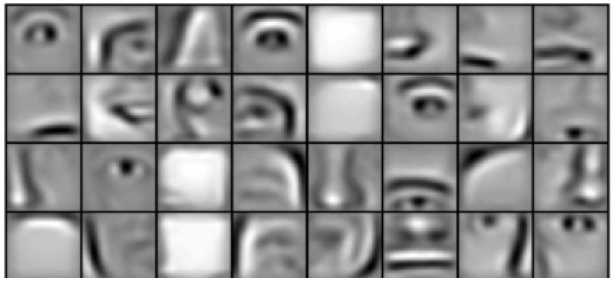
Tabelle: Wortfehlerraten auf zwei Testkorpora

$\Rightarrow$ Reduktion der Fehlerrate um ein Drittel!

Neuronale Netze in der Bilderkennung

Krizhevsky et al. (2012): ImageNet Classification with Deep Convolutional Neural Networks

Ebene 3: Gesichter

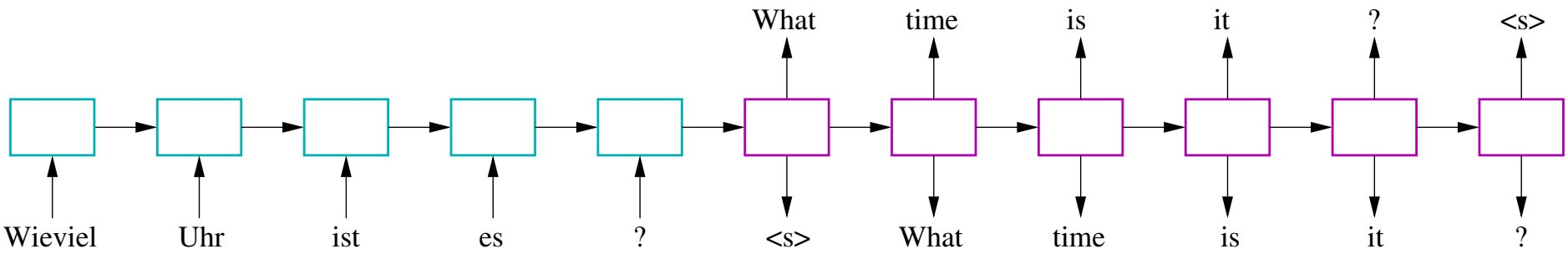


Ebene 2: Augen, Nasen, Münder

Ebene 1: einfache Kanten und Flächen

Neuronale Netze in der maschinellen Übersetzung

Bahdanau et al. (2014): Neural Machine Translation by Jointly Learning to Align and Translate



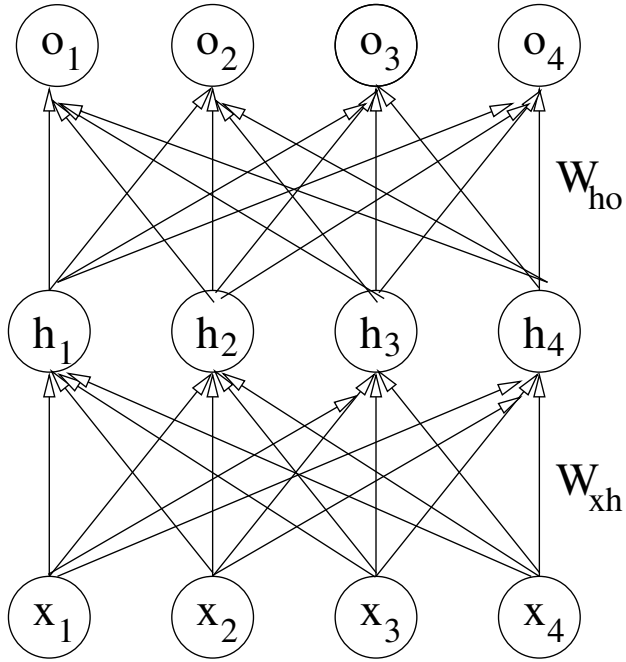
Encoder-Decoder-Architektur mit rekurrenten neuronalen Netzen

Arbeitsweise eines neuronalen Netzes

- Forward-Schritt: Die Aktivierung der Eingabeneuronen wird über die Neuronen der versteckten Ebenen zu den Neuronen der Ausgabeebene propagiert.
- Backward-Schritt: Im Training wird anschließend der Gradient einer zu optimierenden Zielfunktion an den Ausgabeneuronen berechnet und zu den Eingabeneuronen zurückpropagiert.

Forward-Propagation

Ein neuronales Netz mit Eingabevektor $\boldsymbol{x}$, verstecktem Vektor $\boldsymbol{h}$ und skalarem Ausgabewert o kann wie folgt definiert werden:



$$
\begin{aligned}
\boldsymbol{net}_h(\boldsymbol{x}) &= W_{hx}\boldsymbol{x} + \boldsymbol{b}_h \\
\boldsymbol{h}(\boldsymbol{x}) &= \tanh(\boldsymbol{net}_h) \\
&= \tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) \\
\boldsymbol{net}_o(\boldsymbol{x}) &= W_{oh}\boldsymbol{h} + \boldsymbol{b}_o \\
&= W_{oh}\tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o \\
\boldsymbol{o}(\boldsymbol{x}) &= \mathrm{softmax}(\boldsymbol{net}_o) \\
&= \mathrm{softmax}(W_{oh}\tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o)
\end{aligned}
$$

Das Training maximiert die Log-Likelihood der Daten

$$
\begin{aligned}
LL(\boldsymbol{x}, \boldsymbol{y}) &= log(\boldsymbol{y}^T\boldsymbol{o}) \\
&= log(\boldsymbol{y}^T\mathrm{softmax}(W_{oh}\tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o))
\end{aligned}
$$

$\boldsymbol{y}$ repräsentiert die gewünschte Ausgabe als 1-hot-Vektor

Backward-Propagation

Für die Parameteroptimierung müssen die Ableitungen des Ausdruckes

$$LL(\boldsymbol{x}, \boldsymbol{y}) = log(\boldsymbol{y}^T \text{softmax}(W_{oh} \tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o))$$

bzgl. der Parameter W_{hx}, $\boldsymbol{b}_h$, W_{oh} und $\boldsymbol{b}_o$ berechnet werden.

Am besten benutzt man dazu **automatische Differentiation** (z.B. in PyTorch).

Anschließend Anpassung der Parameter θ mit Gradientenanstieg:

$$\theta_{t+1} = \theta_t + \eta \nabla LL_{\theta_t}$$

Das Training mit Gradientenanstieg ist oft langsam.

Es gibt verschiedene Techniken, um es zu beschleunigen. $\rightarrow$

Backward-Propagation

Für die Parameteroptimierung müssen die Ableitungen des Ausdruckes

$$LL(\boldsymbol{x}, \boldsymbol{y}) = log(\boldsymbol{y}^T \text{softmax}(W_{oh} \tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o))$$

bzgl. der Parameter W_{hx}, $\boldsymbol{b}_h$, W_{oh} und $\boldsymbol{b}_o$ berechnet werden.

Am besten benutzt man dazu **automatische Differentiation** (z.B. in PyTorch).

Anschließend Anpassung der Parameter θ mit Gradientenanstieg:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \nabla LL_{\boldsymbol{\theta}_t}$$

Das Training mit Gradientenanstieg ist oft langsam.

Es gibt verschiedene Techniken, um es zu beschleunigen. →

Backward-Propagation

Für die Parameteroptimierung müssen die Ableitungen des Ausdruckes

$$LL(\boldsymbol{x}, \boldsymbol{y}) = log(\boldsymbol{y}^T \text{softmax}(W_{oh} \tanh(W_{hx}\boldsymbol{x} + \boldsymbol{b}_h) + \boldsymbol{b}_o))$$

bzgl. der Parameter W_{hx}, $\boldsymbol{b}_h$, W_{oh} und $\boldsymbol{b}_o$ berechnet werden.

Am besten benutzt man dazu **automatische Differentiation** (z.B. in PyTorch).

Anschließend Anpassung der Parameter θ mit Gradientenanstieg:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta \nabla LL_{\boldsymbol{\theta}_t}$$

Das Training mit Gradientenanstieg ist oft langsam.

Es gibt verschiedene Techniken, um es zu beschleunigen. $\rightarrow$

Beschleunigung des Trainings

- Momentum-Term

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu \boldsymbol{v}_t + \eta \nabla LL_{\theta_t} \quad \text{mit } 0 < \mu < 1 \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned}$$

Verbesserung: Nesterov's Accelerated Gradient:

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu \boldsymbol{v}_t + \eta \partial LL(\boldsymbol{\theta}_t + \mu \boldsymbol{v}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned}$$

Da man sowieso im nächsten Schritt μv_t zu θ_t addieren wird, ist es besser gleich mit $\theta_t + \mu v_t$ zu evaluieren.

- Lernratenoptimierung:
  - Lernrate kleiner werden lassen, z.B. $\eta_t = \eta_{max} \frac{1}{1+\delta t}$
  - gewichtspezifische Lernraten: Adagrad, RMSProp, Adadelta, Adam

Beschleunigung des Trainings

- Momentum-Term

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu \boldsymbol{v}_t + \eta \nabla LL_{\theta_t} \quad \text{mit } 0 < \mu < 1 \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned}$$

  Verbesserung: Nesterov's Accelerated Gradient:

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu \boldsymbol{v}_t + \eta \partial LL(\boldsymbol{\theta}_t + \mu \boldsymbol{v}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned}$$

  Da man sowieso im nächsten Schritt μv_t zu θ_t addieren wird, ist es besser gleich mit $\theta_t + \mu v_t$ zu evaluieren.

- Lernratenoptimierung:
  - Lernrate kleiner werden lassen, z.B. $\eta_t = \eta_{max} \frac{1}{1+\delta t}$
  - gewichtspezifische Lernraten: Adagrad, RMSProp, Adadelta, Adam

Beschleunigung des Trainings

- Momentum-Term

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu \boldsymbol{v}_t + \eta \nabla LL_{\theta_t} \quad \text{mit } 0 < \mu < 1 \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned}$$

  Verbesserung: Nesterov's Accelerated Gradient:

$$\begin{aligned} \boldsymbol{v}_{t+1} &= \mu \boldsymbol{v}_t + \eta \partial LL(\boldsymbol{\theta}_t + \mu \boldsymbol{v}_t) \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1} \end{aligned}$$

  Da man sowieso im nächsten Schritt μv_t zu θ_t addieren wird, ist es besser gleich mit $\theta_t + \mu v_t$ zu evaluieren.

- Lernratenoptimierung:
  - Lernrate kleiner werden lassen, z.B. $\eta_t = \eta_{max} \frac{1}{1+\delta t}$
  - gewichtspezifische Lernraten: Adagrad, RMSProp, Adadelta, Adam

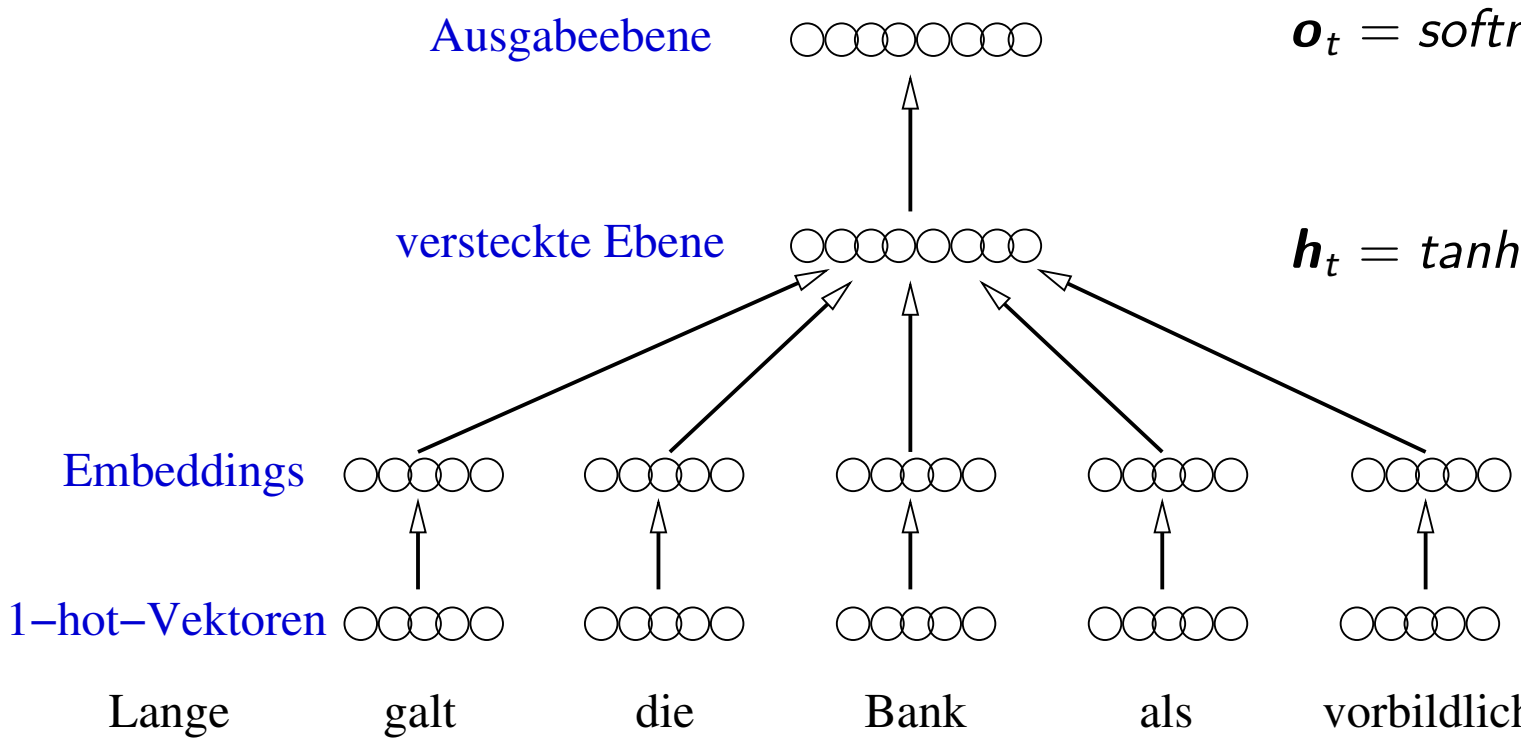
Vor- und Nachteile von NN

+ kein Feature Engineering notwendig
+ gute Ergebnisse ohne lange Entwicklungszeiten
+ geringer Speicherplatzbedarf

– Blackbox: schwer zu analysieren, wie ein Ergebnis zustandekam
– Training ist sehr rechenintensiv
  Lösung: Parallelisierung auf Mehrprozessorsystemen (vor allem GPUs)
– Es gibt sehr viel Metaparameter, an denen man herumschrauben kann
  Architektur, Zahl der Neuronen/Ebenen, Aktivierungsfunktion, Initialisierung, Lernrate usw.

In der maschinellen Sprachverarbeitung sind neuronale Netze dabei, andere Lernverfahren in vielen Bereichen zu verdrängen.

Einfaches neuronales Netz für Wortart-Annotation



$$\boldsymbol{o}_t = softmax(W_{oh}\boldsymbol{h}_t)$$

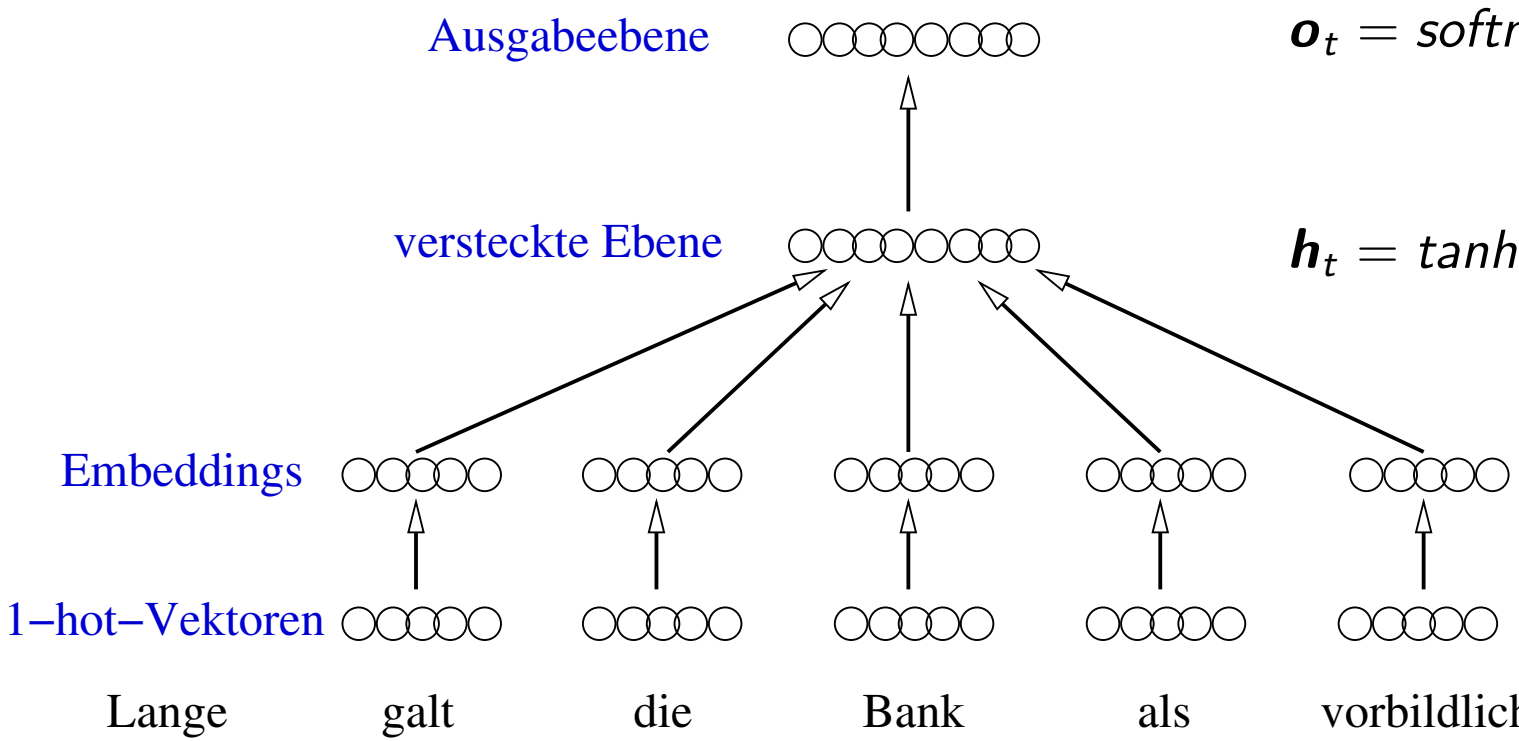
$$\boldsymbol{h}_t = tanh(W_{he}\begin{bmatrix}\boldsymbol{e}_{t-2}\\ \boldsymbol{e}_{t-1}\\ \boldsymbol{e}_t\\ \boldsymbol{e}_{t+1}\\ \boldsymbol{e}_{t+2}\end{bmatrix})$$

$$\boldsymbol{e_t} = L\boldsymbol{w_t}$$

"Feedforward"-Netze haben ein beschränktes Eingabefenster und können daher keine langen Abhängigkeiten erfassen.

→ rekurrente Netze

Einfaches neuronales Netz für Wortart-Annotation



$$\boldsymbol{o}_t = softmax(W_{oh}\boldsymbol{h}_t)$$

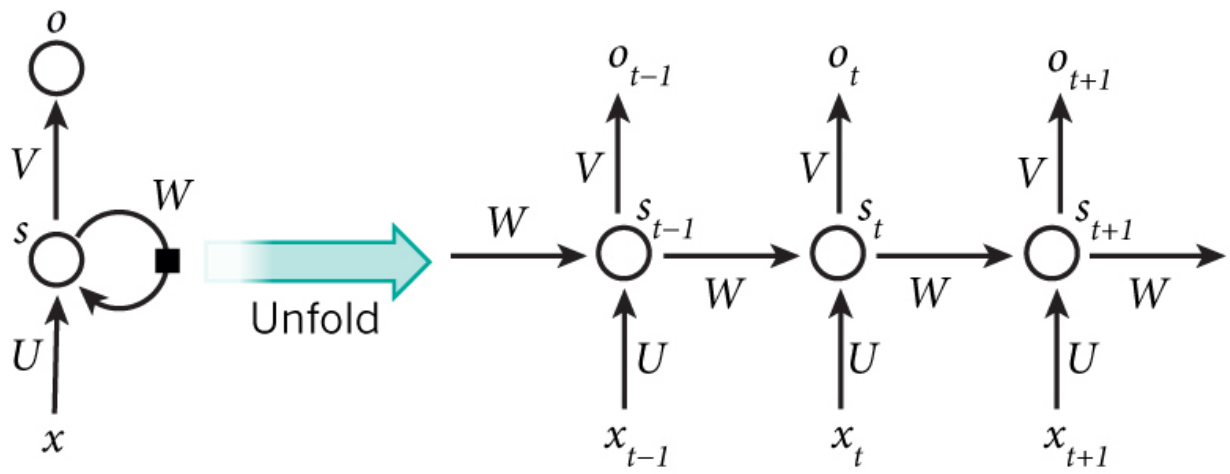
$$\boldsymbol{h}_t = tanh(W_{he}\begin{bmatrix}\boldsymbol{e}_{t-2}\\ \boldsymbol{e}_{t-1}\\ \boldsymbol{e}_t\\ \boldsymbol{e}_{t+1}\\ \boldsymbol{e}_{t+2}\end{bmatrix})$$

$$\boldsymbol{e_t} = L\boldsymbol{w_t}$$

"Feedforward"-Netze haben ein beschränktes Eingabefenster und können daher keine langen Abhängigkeiten erfassen.

→ rekurrente Netze

Rekurrente Neuronale Netze

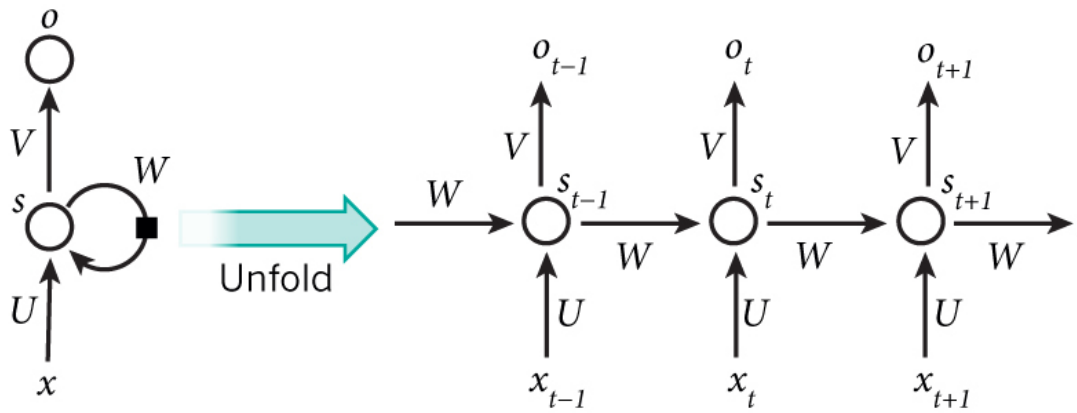


http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns

Ein RNN ist äquivalent zu einem sehr tiefen Feedforward-NN mit gekoppelten Gewichtsmatrizen.

Das neuronale Netz lernt, relevante Information über die Vorgängerwörter im Zustand s_t zu speichern.

Rekurrentes neuronales Netz für Wortart-Annotation

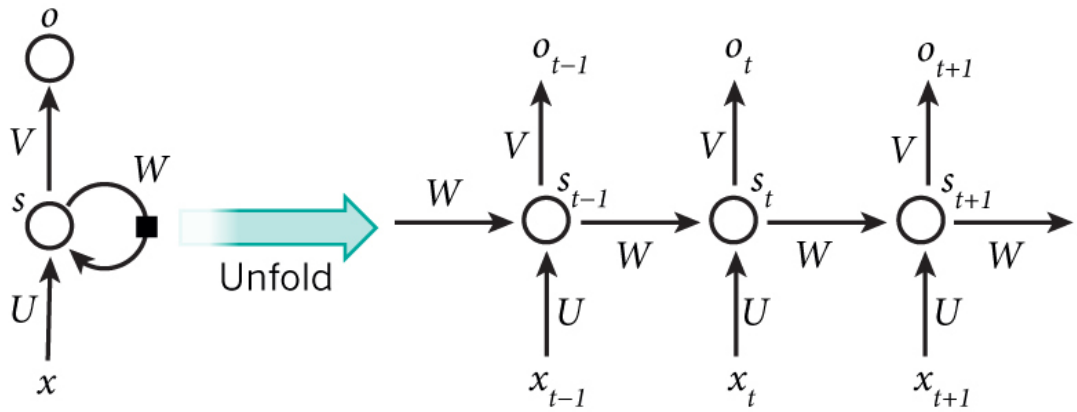


In jedem Schritt

- wird ein Wort x_t gelesen
- ein neuer Hidden State s_t (= Aktivierungen der versteckten Ebene) berechnet
- eine Wahrscheinlichkeitsverteilung über mögliche Tags ausgegeben o_t

(Das Netzwerk hat noch keine Information über den rechten Kontext. Dazu später mehr.)

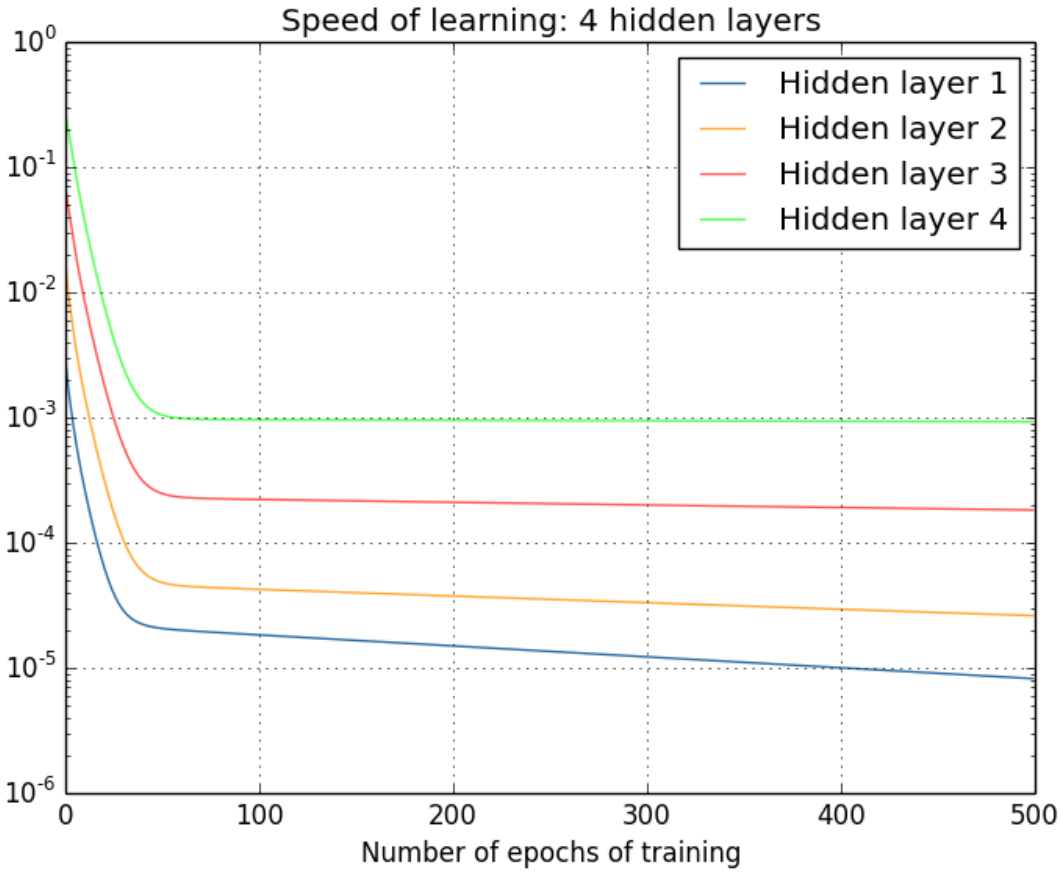
Backpropagation Through Time (BPTT)



- Zum Trainieren wird das rekurrente Netz zu einem Feedforward-Netz aufgefaltet
- Der Gradient wird von den Ausgabeneuronen o_t zu den versteckten Neuronen s_t und von dort weiter zu den Eingabeneuronen x_t und den vorherigen versteckten Neuronen s_{t-1} propagiert (Backpropagation through time)
- An den versteckten Neuronen werden zwei Gradienten addiert.

Verschwindender/Explodierender Gradient

Das Training tiefer neuronaler Netze ist schwierig, weil der Gradient beim Zurückpropagieren meist schnell kleiner wird.



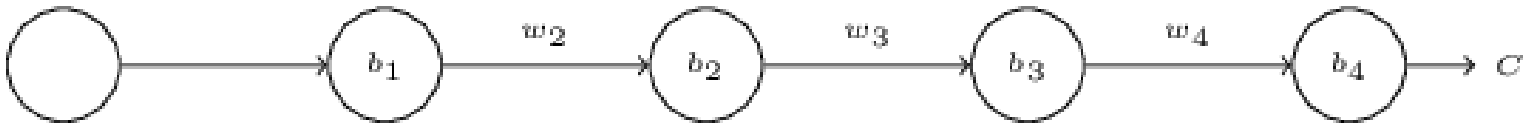
http://neuralnetworksanddeeplearning.com/chap5.html

Verschwindender/Explodierender Gradient

Warum wird der Gradient exponentiell kleiner mit der Zahl der Ebenen?

Betrachten wir ein neuronales Netz mit 5 Ebenen und einem Neuron pro Ebene

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



http://neuralnetworksanddeeplearning.com/chap5.html

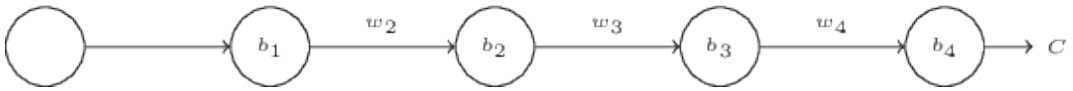
w_i ist ein Gewicht, b_i ein Bias, C die Kostenfunktion, a_i die Aktivierung eines Neurons und z_i die gewichtete Eingabe eines Neurons

Der Gradient wird in jeder Ebene mit dem Ausdruck $w_i \times \sigma'(z_i)$ multipliziert.

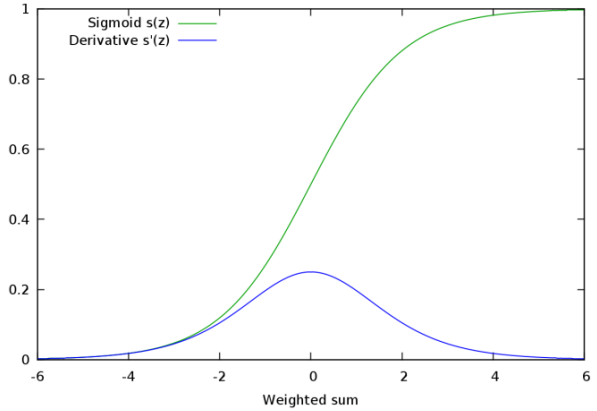
Wie groß ist dieser Ausdruck?

Verschwindender/Explodierender Gradient

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$



http://neuralnetworksanddeeplearning.com/chap5.html



http://whiteboard.ping.se/MachineLearning/BackProp

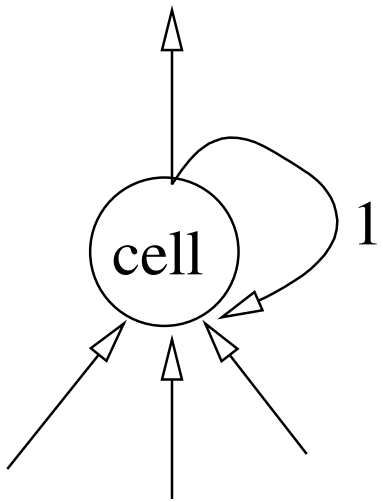
Wenn die Gewichte mit zufälligen Werten im Bereich [-1,1] initialisiert werden, dann gilt $|w_i \times \sigma'(z_i)| < 0.25$, da $|\sigma'(z_i)| < 0.25$.

Damit wird der Gradient exponentiell kleiner mit der Zahl der Ebenen.

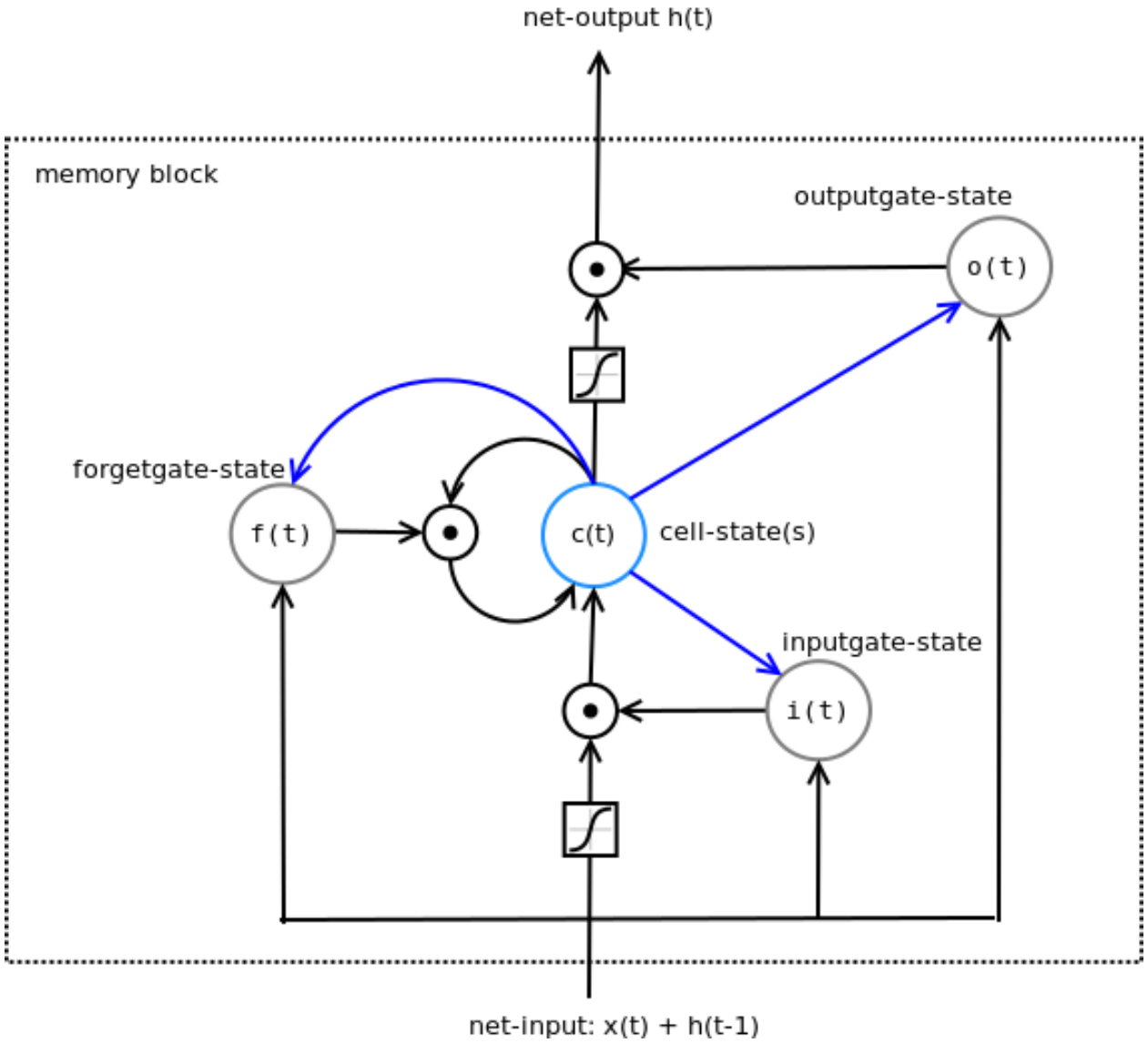
Wenn die Gewichte mit sehr großen Werten initialisiert werden, kann der Gradient auch explodieren.

Long Short Term Memory

- entwickelt von Sepp Hochreiter und Jürgen Schmidhuber (TUM)
- löst das Problem der instabilen Gradienten für rekurrente Netze
- Eine Speicherzelle (cell) bewahrt den Wert des letzten Zeitschritts
- Der Gradient wird beim Zurückpropagieren nicht mehr mit Gewichten multipliziert und bleibt über viele Zeitschritte erhalten



Long Short Term Memory: Schaltplan



http://christianherta.de/lehre/dataScience/machineLearning/neuralNetworks/LSTM.php

Long Short Term Memory

Berechnung eines LSTMs (ohne Peephole-Verbindungen)

$$
\begin{aligned}
z_t &= \tanh(W_z x_t + R_z h_{t-1} + b_z) && \text{(input activation)} \\
i_t &= \sigma(W_i x_t + R_i h_{t-1} + b_i) && \text{(input gate)} \\
f_t &= \sigma(W_f x_t + R_f h_{t-1} + b_f) && \text{(forget gate)} \\
o_t &= \sigma(W_x x_t + R_o h_{t-1} + b_o) && \text{(output gate)} \\
c_t &= f_t \odot c_{t-1} + i_t \odot z_t && \text{(cell)} \\
h_t &= o_t \odot tanh(c_t) && \text{(output activation)}
\end{aligned}
$$

mit $a \odot b = (a_1 b_1, a_2 b_2, ..., a_n b_n)$

Die LSTM-Zellen ersetzen einfache normale Neuronen in rekurrenten Netzen.

Man schreibt kurz:

$$\boldsymbol{z} = LSTM(\boldsymbol{x})$$

Long Short Term Memory

Vorteile

- löst das Problem mit instabilen Gradienten
- exzellente Ergebnisse in vielen Einsatzbereichen

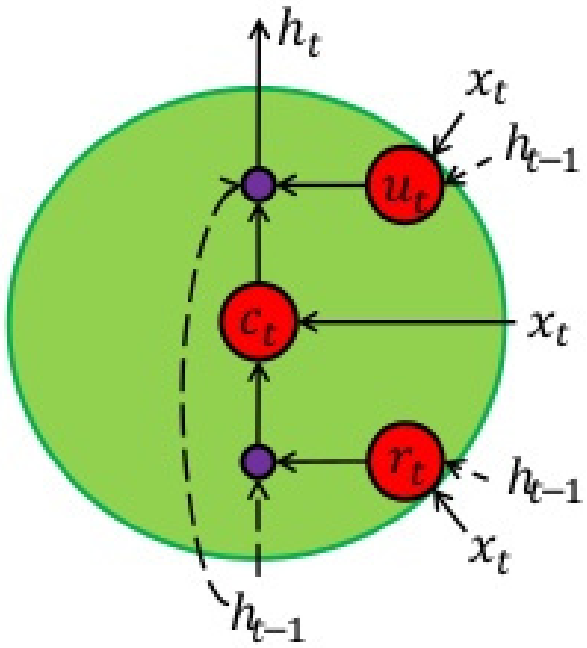
Nachteile

- deutlich komplexer als ein normales Neuron
- höherer Rechenzeitbedarf

Alternative

- Gated recurrent Units (GRU) (Cho et al.)
- etwas einfacher (nur 2 Gates)

Gated Recurrent Units



Gate values in [0,1]

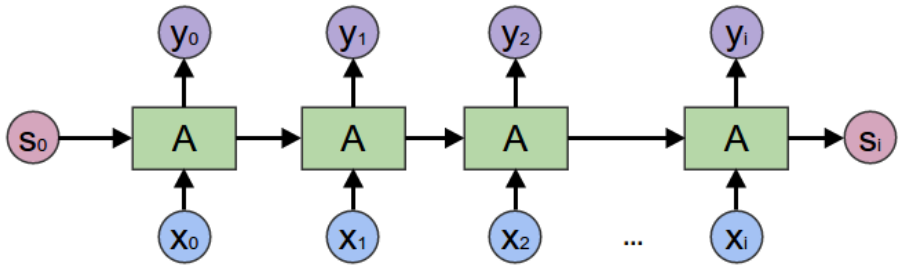
r_t, u_t - reset/update gates

$$u_t = \sigma(V_u x_t + W_u h_{t-1} + b_u)$$
$$r_t = \sigma(V_r x_t + W_r h_{t-1} + b_r)$$

$$c_t = g(V_c x_t + W_c(h_{t-1} \cdot r_t))$$
$$h_t = (1 - u_t) \cdot c_t + u_t \cdot h_{t-1}$$

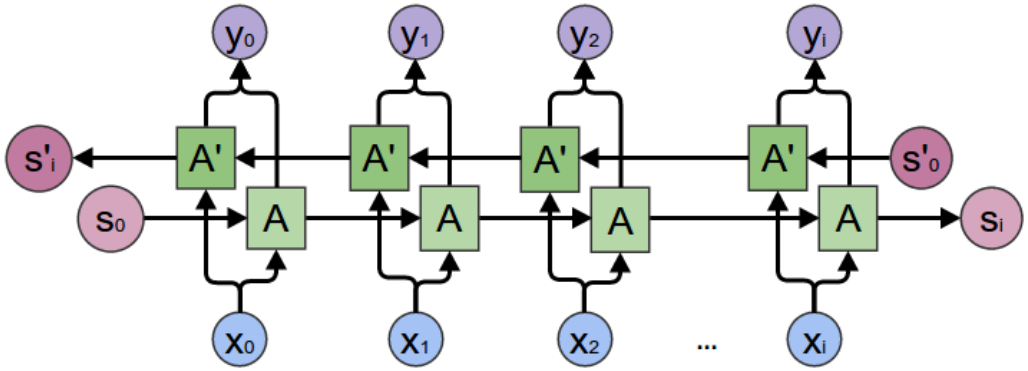
http://image.slidesharecdn.com/2016-clipping-lstm-gru-urnn2851-160727133046/95/2016-clipping-lstm-gru-urnn-12-638.jpg

Bidirektionale RNNs



colah.github.io/posts/2015-09-NN-Types-FP/img/RNN-general

- Die rekurrenten Neuronen repräsentieren alle bisherigen Eingaben.
- Für viele Anwendungen ist aber auch eine Repräsentation der folgenden Eingaben nützlich. ⇒ bidirektionales RNN



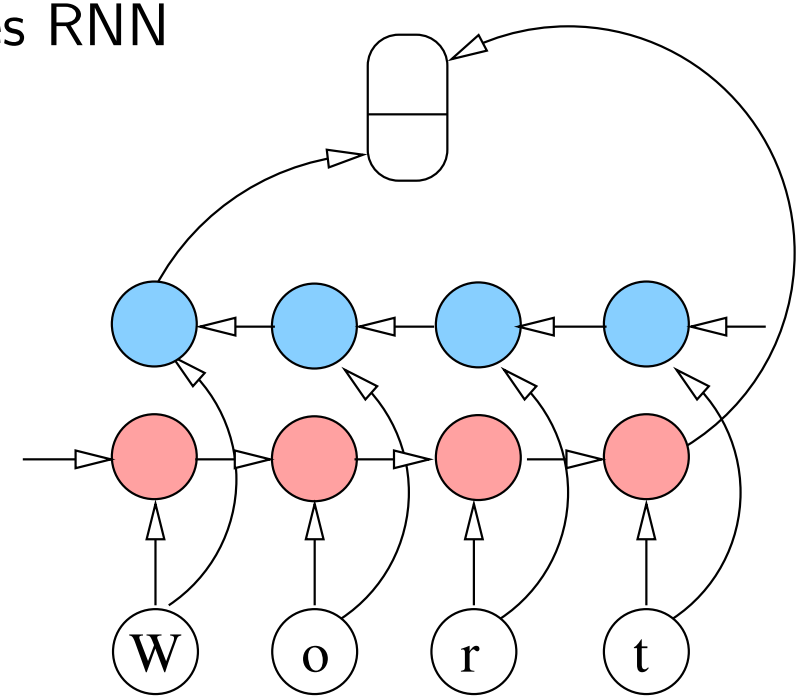
colah.github.io/posts/2015-09-NN-Types-FP/img/RNN-bidirectional

- Bidirektionale RNNs können zu tiefen bidirektionalen RNNs gestapelt werden

Buchstabenbasierte Embeddings

- Die bisher betrachteten Embeddings lernen nichts über die Form des Wortes (= Buchstabenfolge)
  Nichterweislichkeit, Nebengottesdienste, Lepidolith
- Diese Form liefert aber vor allem bei seltenen und unbekannten Wörtern wichtige Informationen.

$\Rightarrow$ Berechnung der Wortrepräsentationen durch ein bidirektionales Buchstaben-basiertes RNN



Programmierung von neuronalen Netzen

Verbreitete Frameworks

- Theano + Python (Universität Montreal)
- Tensorflow + Python (Google)
- Keras (nutzt Theano oder Tensorflow als Backend)
- Torch + Lua (IDIAP)
- **PyTorch** + Python (Facebook)

Programmierung von neuronalen Netzen

PyTorch

- Python-Bibliothek für neuronale Netze und andere komplexe mathematische Funktionen
- Syntax ähnelt **Numpy**
- automatisches Differenzieren von Funktionen ($\rightarrow$ NN-Training)
- GPU-Unterstützung

- Vorteile gegenüber Theano/Tensorflow/Keras
  - PyTorch-Befehle werden sofort ausgeführt (ohne erst einen Berechnungsgraphen erstellen zu müssen)
  - normale Programmierung
  - einfaches Debugging

Numpy

- Python-Bibliothek für numerische Aufgaben
- Datentypen: Skalare, Vektoren, Matrizen und Tensoren
- viele Operatoren (z.B. Matrizenprodukte)
- effiziente C-Implementierungen der Operatoren
- Numpy+SciPy+Matplotlib kann MATLAB ersetzen

Numpy

wichtigste Datenstruktur: ndarray (homogenes mehrdimensionales Array)

```
>>> import numpy as np
>>> a = np.array([(1,2,3),(4,5,6)])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a.shape
(2, 3)
>>> a.ndim
2
>>> a.size
6
>>> type(a)
<type 'numpy.ndarray'>
>>> a.dtype.name
'int64'
```

Numpy

Arrayerzeugung

```
>>> np.array((2, 3, 4))
array([2, 3, 4])
>>> np.array([2, 3, 4.0])
array([2., 3., 4.])
>>> np.array([(1,2),(3,4)], dtype=float)
array([[ 1.,  2.],
       [ 3.,  4.]])
>>> np.zeros((2,3))
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
>>> np.ones((2,3))
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> np.full(3,0.1)
array([ 0.1,  0.1,  0.1])
```

Numpy

Zahlenfolgen

```
>>> np.arange(3)
array([0, 1, 2])
>>> np.arange(3.0)
array([0., 1., 2.])
>>> np.arange(3,0,-1)
array([3, 2, 1])
>>> np.arange(3,0,-.9)
array([ 3., 2.1, 1.2, 0.3])
>>> np.arange(6).reshape(2,3)
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.linspace(0,2,9)
array([ 0., 0.25, 0.5, 0.75, 1., 1.25, 1.5, 1.75, 2. ])
```

Numpy

Arrayerzeugung (Cont'd)

```
>>> a = np.arange(6).reshape(2,3)
>>> a
array([[0, 1, 2],
       [3, 4, 5]])
>>> b = np.ones_like(a, dtype=float)
>>> b
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> np.identity(3)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

Numpy

Arithmetische Operationen

```
>>> a = np.array([20,30,40,50])
>>> b = np.arange( 4 )
>>> b-2
array([20, 29, 38, 47])
>>> a*b
array([  0,  30,  80, 150])
>>> b**2
array([0, 1, 4, 9])
>>> 10 * np.sin(a)
array([ 9.12945251, -9.88031624,  7.4511316 , -2.62374854])
>>> a<35
array([ True,  True, False, False], dtype=bool)
```

Numpy

Matrix-Operationen

```
>>> a
array([20, 30, 40, 50])
>>> b
array([0, 1, 2, 3])
>>> a.dot(b)
260
>>> M = np.arange(8).reshape(2,4)
>>> M
array([[0, 1, 2, 3],
       [4, 5, 6, 7]])
>>> M.dot(a)
array([260, 820])
```

Numpy

Matrix-Operationen

```
>>> a.dot(M)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: objects are not aligned
>>> M.transpose()    // = M.T
array([[0, 4],
       [1, 5],
       [2, 6],
       [3, 7]])
>>> a.dot(M.transpose())
array([260, 820])
```

Numpy

Matrix-Operationen

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.88924321,  0.18706006,  0.08759729],
       [ 0.45328557,  0.49605247,  0.12153046]])
>>> a.sum()
2.2347690651454872
>>> a.min()
0.087597291287608847
>>> a.max(axis=0)
array([ 0.88924321,  0.49605247,  0.12153046])
>>> a.max(axis=1)
array([ 0.88924321,  0.49605247])
>>> a.argmax(axis=1)
array([0, 1])
```

Numpy

Broadcasting

```
>>> x1
array([[ 0.,  1.,  2.],
       [ 3.,  4.,  5.],
       [ 6.,  7.,  8.]])
>>> x2
array([ 0.,  1.,  2.])
>>> x1+x2
array([[  0.,   2.,   4.],
       [  3.,   5.,   7.],
       [  6.,   8.,  10.]])
>>> np.add(x1, x2)
array([[  0.,   2.,   4.],
       [  3.,   5.,   7.],
       [  6.,   8.,  10.]])
```

Numpy

Slicing

```
>>> a = np.arange(10)**3
>>> a
array([  0,   1,   8,  27,  64, 125, 216, 343, 512, 729])
>>> a[2]
8
>>> a[2:5]
array([ 8, 27, 64])
>>> a[0:6:2]
array([ 0,  8, 64])
>>> a[:6:2]=-1
>>> a
array([ -1,   1,  -1,  27,  -1, 125, 216, 343, 512, 729])
>>> a[::-1]
array([729, 512, 343, 216, 125,  -1,  27,  -1,   1,  -1])
```

Numpy

Multidimensional Slicing

```
>>> def f(x,y):
...     return 10*x+y
>>> b = np.fromfunction(f,(5,4),dtype=int)
>>> b
array([[ 0,  1,  2,  3],
       [10, 11, 12, 13],
       [20, 21, 22, 23],
       [30, 31, 32, 33],
       [40, 41, 42, 43]])
>>> b[2,3]
23
>>> b[0:5, 1]
array([ 1, 11, 21, 31, 41])
>>> b[:, 1]
array([ 1, 11, 21, 31, 41])
>>> b[1:3, 2:0:-1]
array([[12, 11],
       [22, 21]])
```

Numpy

Advanced Indexing

```
>>> a = numpy.arange(12).reshape(3,4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a[[(0,1),(1,2)]]
array([1, 6])
>>> a[[0,2]]
array([[ 0,  1,  2,  3],
       [ 8,  9, 10, 11]])
>>> a[a>5]
array([ 6,  7,  8,  9, 10, 11])
```

Numpy

Array Reshaping

```
>>> a = np.floor(10*np.random.random((3,4)))
>>> a.shape
(3, 4)
>>> a.flatten()   // liefert eine Kopie
array([ 9., 6., 5., 7., 9., 1., 6., 1., 3., 2., 4., 2.])
>>> a.ravel()     // liefert eine Ansicht
array([ 9., 6., 5., 7., 9., 1., 6., 1., 3., 2., 4., 2.])
>>> a.shape = (6,2)
>>> a
array([[ 9.,  6.],
       [ 5.,  7.],
       [ 9.,  1.],
       [ 6.,  1.],
       [ 3.,  2.],
       [ 4.,  2.]])
>>> a.T
array([[ 9.,  5.,  9.,  6.,  3.,  4.],
       [ 6.,  7.,  1.,  1.,  2.,  2.]])
```

Numpy

Views

```
>>> a = np.arange(12)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> b = a.view()
>>> b
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> b.resize((3,4))     // wie reshape aber "in-place"
>>> b
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

PyTorch

PyTorch-Beispiel

```
import functools, operator, torch
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input channel, 6 output channels, 5x5 convolution kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
    ...
```

PyTorch-Beispiel (Forts.)

```
class Net(nn.Module):
    ...

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square you can only specify a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)

        num_flat_features = self.prod(x.size()[1:])
        x = x.view(-1, num_flat_features)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

    def prod(self, x):  # Python has no prod function:
        return functools.reduce(operator.mul, x, 1)
    ...
```

PyTorch-Beispiel (Forts.)

```
...
net = Net()  # build the network
criterion = nn.MSELoss()  # definition of the loss function
optimizer = optim.SGD(net.parameters(), lr=0.01) # optimizer

input  = Variable(torch.randn(1, 1, 32, 32)) # some random input
output = net(input) # calls the forward method

target = Variable(torch.arange(1, 11))  # a dummy target, for example
target = target.view(1, -1)  # add a batch dimension

net.zero_grad()  # do not accumulate gradients
loss = criterion(output, target)
loss.backward()  # backpropagation
optimizer.step() # weight update
```

Batchverarbeitung

- Die vordefinierten PyTorch-NN-Module erwarten eine Folge von Trainingsbeispielen (Minibatch) als Eingabe.
- Die Minibatch-Verarbeitung lastet die GPU besser aus.
- Wenn auf Sätzen trainiert wird, müssen alle Sätze **dieselbe Länge** haben, damit sie zu einer Matrix zusammengefasst werden können.
- Kürzere Sätze müssen daher mit Dummysymbolen aufgefüllt werden (Padding)

Konstituentenparsing mit neuronalen Netzen

Grundlage: Gaddy et al. (Link auf Kursseite)

Strategie:

- Berechne für jede mögliche Konstituente (i, k, l) mit Startposition i, Endposition k und Kategorie l eine Bewertung $s(i, k, l)$.
- Die Bewertung eines Parsebaumes T ist die Summe der Bewertungen aller enthaltenen Konstituenten:

$$s(T) = \sum_{(i,k,l) \in T} s(i, k, l)$$

- Der Parser sucht den am besten bewerteten Parse:

$$\hat{T} = \arg\max_{T} s(T)$$

Anm.: Die Labels der Tochter-Konstituenten haben keinen Einfluss auf den Score $s(i, k, l)$ einer Konstituente.

Konstituentenparsing mit neuronalen Netzen

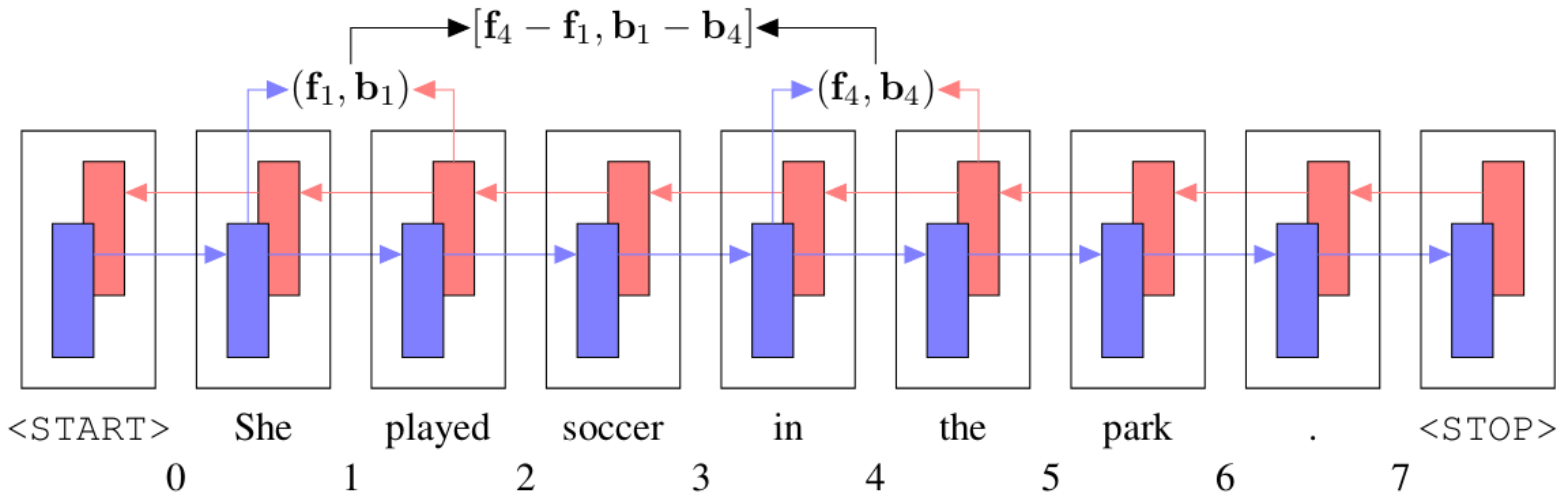
Zu klärende Fragen

- Wie werden mögliche Konstituenten repräsentiert?
- Wie werden die Konstituenten-Kategorien bewertet?
- Wie werden Wörter repräsentiert?
- Wie wird der beste Parse berechnet?
- Wie wird der Parser trainiert?

Repräsentation der Konstituenten

- Ein BiLSTM berechnet eine Vorwärts-Repräsentation f_i und eine Rückwärts-Repräsentation b_i für jede Satzposition i.
- Die Repräsentation r_{ik} einer Konstituente (i, k) wird als Konkatenation zweier Differenzvektoren definiert:

$$\mathbf{r}_{ik} = [\mathbf{f}_k - \mathbf{f}_i, \mathbf{b}_i - \mathbf{b}_k]$$



Bewertung der Konstituenten-Kategorien

Die Bewertungen der Kategorien werden durch ein neuronales Netz mit einer Hidden Layer aus der Konstituenten-Repräsentation berechnet:

$$s(i,k,l) = [\mathbf{W}_2 g(\mathbf{W}_1 \mathbf{r}_{ik} + z_1) + z_2]_l$$

g ist hier die Aktivierungsfunktion, z.B. ReLU.

Die Ausgabeebene enthält für jede Konstituenten-Kategorie ein Neuron und verwendet keine Aktivierungsfunktion.

Repräsentation der Wörter

- **Möglichkeit 1:** Wort-Embeddings

  Nachteil: schlechte Repräsentationen für seltene/unbekannte Wörter

- **Möglichkeit 2:** buchstabenbasierte Repräsentation berechnet mit BiLSTM

  Endzustände des Vorwärts- und Rückwärts-LSTMs werden konkateniert.

- Die beiden Repräsentationen können auch **kombiniert** werden $\Rightarrow$

  Gaddy et al. zeigen aber, dass buchstabenbasierte Repräsentationen ausreichen.

Parsing

- verwendet den CKY-Algorithmus
- Kettenregeln der Form VP $\rightarrow$ V werden durch Verwendung komplexer Kategorien VP-V eliminiert.
- Nicht-binäre Konstituenten wie VP $\rightarrow$ V NP PP werden geparst, indem zunächst V und NP zu einer Hilfskonstituente der Kategorie $\emptyset$ zusammengefasst werden, wobei $s(i, k, \emptyset) = 0$.

Parsing

- Berechnung der Viterbi-Scores von Konstituenten der Länge 1

$$\delta(i, i+1) = \max_l s(i, i+1, l)$$

= Bewertung der besten Kategorie l

- Berechnung der Viterbi-Scores von Konstituenten der Länge $k - i > 1$

$$\delta(i, k) = \max_l s(i, k, l) + \max_{i<j<k} [\delta(i, j) + \delta(j, k)]$$

Wichtig: Die beste Kategorie l und die beste binäre Zerlegung der Konstituente können unabhängig voneinander berechnet werden.

Parsing

- Der Parser berechnet bottom-up alle Viterbi-Scores $\delta(i, k)$.
- Er merkt sich dabei für jede (mögliche) Konstituente die beste Kategorie und die beste binäre Zerlegung.
- Schließlich wird der beste Parsebaum top-down extrahiert.

Pseudocode

compute $score[i, k, sym]$ for all i, k, sym # Bewertungen aller möglichen Konstituenten
for l in 1...n **do** # Für alle Konstituentenlängen
    for i in 0...n-l **do** # Für alle Startpositionen
        k = i+l # Endposition
        $vscore[i, k] = \max_{sym} score[i, k, sym]$ # Beste Bewertung
        $label[i, k] = \arg\max_{sym} score[i, k, sym]$ # Beste Kategorie
        if l > 1 **then**
            $split[i, k] = \arg\max_{i<j<k} vscore[i, j] + vscore[j, k]$ # Beste Zerlegung
            $vscore[i, k] += vscore[i, split[i, k]] + vscore[split[i, k], k]$
output(0, n) # Parsebaum ausgeben

def output(i,k)
    print "(", label[i,k]
      # Kettenregeln und Dummy-Konstituenten sind hier nicht berücksichtigt
    if k == i+1 **then**
        print word[i]
    else
        output(i, split[i,k]); output(split[i,k], k) # Rekursion
    print ")"

Parser-Training

Der Original-Parser verwendet Margin-basiertes Training mit der Hinge-Lossfunktion

$$max\,(0, max_T[s(T) + \Delta(T, T^*)] - s(T^*))$$

Hier ist $\Delta(T, T^*)$ die Zahl der falsch gelabelten Konstituenten in T.

Das Loss wird 0, wenn die Bewertung $s(T^*)$ des korrekten Parsebaumes T^* um $\Delta(T, T^*)$ größer ist als die Bewertung jedes anderen Parsebaumes T.

Parser-Training

Wir werden stattdessen einfach die Summe der logarithmierten Wahrscheinlichkeiten der korrekten Labels für alle möglichen Konstituenten (i,k) maximieren:

$$\sum_{i=0}^{n-1} \sum_{k=i+1}^{n} \log p_{i,k,\text{label}[i,k]}$$

label$[i,k]$ ist gleich der ID der korrekten Kategorie, falls $w_i, ..., w_{k-1}$ eine Konstituente ist, und andernfalls gleich der ID von $\emptyset$.

$p_{i,k,l} = [\text{softmax}(\text{score}_{i.k})]_l$ ist die Wahrscheinlichkeit der Kategorie mit der ID l für die Wortfolge $w_i, ..., w_{k-1}$.