Künstliche Intelligenz Vorlesung SS 23

Johannes Waldmann, HTWK Leipzig

6. Juli 2023

Kompression = Abstraktion

- Welt: physikalische Größen (Schalldruck, Helligkeit, ...)
- direkte symbolische Repräsentation: Zahlen/Bitfolgen
- effizientere Repräsentation mit Hilfe von Modellen
  - Modell: (wiederholte) Teilfolge,
    Realisierung: Wörterbuch-Kompression (Lempel-Ziv)
  - Modell: Bewegung (eines Teilbildes),
    Realisierung: Video-Codec H264 u.ä.
- Baustein der Modellierung ist: Abstraktion. Dabei:
  - Informationsverlust (Bsp: nicht exakte Wiederholung)
  - Erkenntnisgewinn (Bsp: Bewegungserkennung)
- kann benutzt werden zur Vorhersage des Weltverhaltens

Kompression = Abstraktion = KI?

- ... kann man alles machen, aber: ist das Intelligenz?
- die blanke Erkennung von erwarteten Mustern (das Anwenden von Theorien) wohl nicht (das ist reine Such- oder Optimierungsaufgabe)
- „intelligent“ (im Sinne von: menschlich, statt: mechanisch)
  ist das Problemlösen *ohne* Theorie (Improvisieren)?
  das Erfinden *neuer* Theorien (Abstraktionen, Modelle)

- aber „neu“ läßt sich (in der Anwendung) schwer prüfen: auch die neueste Theorie ergibt schließlich einen Mathematik- oder Programmtext, solche kann man auch aufzählen, würfeln oder geschickt raten.
- zum Prüfen der Theorie braucht man unabhängige (!) Bewertungsfunktion, oft ist das die eigentliche Erfindung.

Beispiel: Hausaufgabenbetrug

- gesucht ist ein Programm, das jede autotool-Aufgabe lösen kann (Bsp: konvertiere NFA zu äquival. Regexp)
- nur durch Verarbeitung der Fehlermeldungen
- ohne eingebaute aufgabenspezifische Theorien (Algorithmen)
- also wie ein Student, der die Vorlesung ignoriert,

  aber mit viel Energie Hausaufgaben bearbeitet
- angenommen, das geht (wenigstens teilweise)—

  kann so ein Programm ein Bachelor-Diplom erhalten? D.h., dann einen Software-Ingenieur ersetzen?
- . . . der weitere autotool-Aufgabentypen entwerfen soll?

Die derzeitige „Definition“ von KI

- . . . Already, China has one of the biggest clusters of AI scientists. It has over 800m internet users, more than any other country, which means *more data on which to hone its new AI*.

  (The Economist 17. 3. 2018, *America vs. China*)
- Bezeichung „KI“ verschleiert tatsächliche wirtschafts-, staats- und geopolitische Interessen und Akteure
- Christian Heck, Hans-Jörg Kreowski (Hrsg.) 2022 `https://blog.fiff.de/online-hearing-kr`
- Philips et al.: *Four Principles of Explainable AI*, NIST 2020 `https://doi.org/10.6028/NIST.IR.8312-draft`

KI als Mode-Erscheinung in der Informatik

- KI = maschinelles Lernen = „deep learning“

  = das Bestimmen von Koeffizienten für konvolutionale neuronale Netze mit mehreren Schichten durch Gradientenabstieg, Bestimmung des Gradienten durch *automatic differentiation*

- ist ein spezielles stochastisches Optimierungsverfahren.

- die dabei bestimmten („gelernten“) Koeffizienten sind eine komprimierte Darstellung der Trainingsmenge.

- das kann sinnvoll sein, solange es keine besseren Darstellungen gibt (das ist die eigentliche Fachfrage)

- aber wenn es dafür derzeit (privates und staatliches) Geld gibt, dann kommt es nicht auf den Sinn an

Das ist nicht die erste KI-Modewelle

- Fifth Generation Computer Systems (staatliche Forschungsförderung in Japan 1982 – 1993)

  Ziel: Soft- und Hardware für massiv parallele Wissensverarbeitung

  `https://web.archive.org/web/20090217105259/http://www.icot.or.jp/ARCHIVE/Museum/ICOT/FGCS-E.html`

- und ähnliche Projekte in USA und Europa

  (wegen FOMO - fear of missing out)

Sachliche Grundlagen der KI

- formale Logik
  (Aristoteles, Leibniz, Frege, Russel, Gödel, Turing, . . .)
  - Syntax (Formeln), Semantik (Wahrheit, Modelle)
  - Kalkül (Axiome und Regeln zum Schließen)
- künstliche neuronale Netze (Warren McCulloch, Walter Pitts, 1943), Gradienten-Abstieg (Newton 1669)

- symbolische Informationsdarstellung (d.h. Daten sind Term-Bäume) und -verarbeitung in Programmiersprache LISP (John McCarthy, 1958)
- Resolutions-Kalkül als operationale Semantik in Programmiersprache Prolog (Alain Colmerauer, 1972)
- Zusammenfassung: wenn es funktioniert, dann ist es keine KI mehr, sondern Wissenschaft

Was machen wir dann in dieser VL?

- für die funktionierenden Verfahren gibt es bereits eigene Vorlesungen (Bsp: Constraint-Programmierung, Symbolisches Rechnen, Numerik/Optimierung)
- wir betrachten an einigen Beispielen, auf denen „KI“ draufsteht, was wirklich drin ist. Aus diesen Gebieten:
  - Audio-Kompression, -Merkmals-Erkennung
  - automatisches Beweisen (in Gleichungstheorien)
  - Spielbaumbewertung (insbesondere für Go)
  - Text-Kompression und -Rekombination (chatgpt)
- wir werden feststellen: diese Verfahren beruhen auf solider Theorie, sind gelegentlich (sehr) erfolgreich
- ob diese automatischen Verfahren tatsächlich Theorien wiederentdecken oder neuerfinden—bleibt offen

Organisatorisches

- pro Woche ein VL, eine Ü
- VL-Skript auf Webseite (nach und nach) `https://www.imn.htwk-leipzig.de/~waldmann/lehre.html`
- Literatur: (kein Lehrbuch)

  Original-Artikel (werden fallweise angegeben)
- Aufgaben:
  - (in Gruppen) Vorrechnen in der nächsten Übung an der Tafel bzw. am Computer

  - (individuell) autotool

- Klausur (120 min, keine Hilfsmittel)

Bei-Spiele für Aufgaben der KI

- wir benutzen oft tatsächlich *Spiele*,
- Einpersonenspiele (*puzzle*)
  - ergänze unvollständige Information (Sudoku, Minesweeper)
    (Spielesteine/Zahlen hinzufügen)
  - finde Zugfolge zu einem bestimmten Zustand (Lunar Lockout, Sokoban) (Spielsteine bewegen)
- Zweipersonenspiele (*game*) (Nim, Gomoku, Go)
  - erzwinge das Erreichen eines bestimmten Zustandes trotz gegnerischer Züge
- das modelliert reale Aufgaben (Planung/Optimierung unter Unsicherheit, mit Gegner)

Nim

- endliches 2-Personenspiel mit vollständiger Information
- Konfiguration: Multimenge M von natürlichen Zahlen
- Zug: ein $x \in M$ durch ein y mit $0 \leq y < x$ ersetzen
- verloren hat, wer nicht mehr ziehen kann
- ist *neutrales* Spiel: mögliche Züge hängen nicht vom Spieler ab. Gegensatz: z.B. Schach, der erste Spieler darf nur weiße Figuren führen.
- Nim hat vollständige mathematische Beschreibung, Status und optimaler nächster Zug können ohne Spielbaum bestimmt werden

Go

- Go (japanisch), Weiqi (chinesisch), Baduk (koreanisch):
  eines der ältesten und verbreitetsten Brettspiele
- Spezifikation:
  - Konfiguration: $\{1 \dots 19\}^2 \to \{\text{weiß,schwarz,leer}\}$
  - Zug: Stein eigener Farbe setzen, gegnerische Steine ohne *Freiheiten* (Gefangene) entfernen
  - keine Konfigurations-Wiederholung
  - Ziel: möglichst großes beherrschtes *Gebiet* (leere Felder, auf welche der Gegner nicht setzen möchte, da er dort gefangen würde)
- Testfall für KI, nachdem Schach prinzipiell gelöst war.
- Seinsei's Library `https://senseis.xmp.net/`, Deutscher Go-Bund `https://dgob.de/`

Aufgaben

1. Emacs: M-x doctor.

   (M-x: gesprochen: Meta x, getippt: ESC (tippen, nicht halten) x)

   Wann, von wem, warum wurde die ursprüngliche Version dieses Programm geschrieben?

   Welche Einschätzung gibt der Autor selbst? Hinweis `https://web.archive.org/web/20050826094312/http://www.gslis.utexas.edu/~palmquis/courses/reviews/amy.htm`

   Finden und lesen Sie den Quelltext (tatsächlich benutzte Version unter `/usr/share/emacs`, Repository: `https://git.savannah.gnu.org/cgit/emacs.git`)

2. Nim: geben Sie den vollständigen Spielbaum an (besser: Darstellung als DAG) für die Startpositionen (Multimengen) $\{2, 2\}$, $\{1, 2, 3\}$.

   Wie kann der erste Spieler in $\{2, 3, 4\}$ gewinnen? (Dazu braucht man nicht den kompletten Baum.)

3. Gomoku

   - Ziel: 5 in einer Reihe (waagerecht, senkrecht, diagonal)

- (Brett und Steine wie Go, aber sonst keine Beziehung)
- Gewinnen Sie gegen M-x gomoku? Auch, wenn der Computer beginnt?
- Diese „KI“ verwendet *keine* Spielbaumsuche, sonder nur eine einfache Heuristik zur Stellungsbewertung.
  Suchen Sie diese im Quelltext.
- Lösen Sie Diagramm 3b, Diagramm 4 aus: Allis et al.: *Go-Moku and Threat Space Search*, 1993 `https://www.mimuw.edu.pl/~awojna/SID/referaty/Go-Moku.pdf`

4. Go

   im Pool installiert sind:

   - `q5go` (Bedienoberfläche),
   - `gnugo`: Daniel Bump, Gunnar Farneback and Arend Bayer; ca. 2010, Go-Programm mit von Hand gebauter Musterbibliothek und klassischer Spielbaumbewertung, `https://www.gnu.org/software/gnugo/gnugo_toc.html`.
   - `katago`: David J. Wu, ca. 2020, Go-Programm mit stochastischer Spielbaumbewertung, dabei Positionsbewertung und Zugvorschläge durch neuronale Netze, `https://github.com/lightvector/KataGo` `https://arxiv.org/abs/1902.10565`.

   zur Benutzung: allgmein: `PATH`, `LD_LIBRARY_PATH` richtig setzen, vgl. `https://www.imn.htwk-leipzig.de/~waldmann/etc/pool`, speziell:

   - q5go: starten und in Preferences/Computer Go eintragen:
     - Name: gnugo, Executable: `gnugo`, Arguments: `--mode gtp --level 12`
     - Name: katago, Executable: `katago`, Arguments: `gtp`, use for analysis.

     Im Fenster *q5goClient*: File-Menu: Play with program,

     - Spielen Sie (Schwarz) gegen gnugo (Weiß), auf 9×9, mit Vorgaben (Handicap)
       Taktische Übungen:
       * fangen Sie irgendeinen Stein (eine Kette) des Gegners,
       * lassen Sie sich nicht fangen,
       * verhindern Sie, daß der Gegner lebt
       * bleiben Sie selbst am Leben

  - opfern Sie eigene Steine, um an anderer Stelle Gewinn zu machen

  Vorsicht: diese Ziele widersprechen sich. Solche lokalen taktischen Aufgaben sind nützlich als Bausteine einer globalen Strategie, um das eigentliche Spielziel (mehr Gebiet) zu erreichen.
  Vorsicht: zum Leben braucht man zwei Augen oder Seki, beim Spielen gibt es evtl. Ko.
  Diese Begriffe und ihre Anwendung *folgen aus den Regeln*, sind aber am Anfang nicht offensichtlich. Siehe Tutorial.

- Computer vs. computer play: gnugo gegen katago. Vorgaben variieren.
- Learn Go ⇒ Tutorials ⇒ Rules, Life and Death, Tactics
  Zeigen Sie eine Beispiel-Aufgabe mit einer Treppe (engl. *ladder*). Was bedeutet die Existenz von Treppen für die (automatisierte) Bewertung von Spielsituationen durch lokale Mustererkennung?

1 Ergänzung zu Spielen (VL KW 15)

Neutrale Spiele

- Positionen eines neutralen Spieles werden bewertet mit:
  N: next player wins, P: previous player wins.
- für ein Spiel mit Zug-Relation $(\to)$:
  $\forall x : x \in N \iff \exists y : (x \to y) \wedge y \in P$
  $\forall x : x \in P \iff \forall y : (x \to y) \Rightarrow y \in N$
- Implementierung (siehe `ki-ss23/nim`)

```
status x =
  if any (== P) (map status $ next x) then N else P
```

  mit Memoisierung `status = memoFix $ \ st x -> ...`

  `https://hackage.haskell.org/package/memoize-1.1.2`

Nim und Schiebe-Nim

- für alle zweistelligen Nim-Positionen (Multimengen) gilt: $\{a, b\} \in P \iff a = b$.
  Beweis: benutze Definition von P/N, Induktion über $a + b$.

- Schiebe-Nim: Positionen $=$ streng monoton steigende Folgen natürlicher Zahlen,

  Züge: $\ldots, x_k, \cdots \to \ldots, x'_k, \ldots$ mit $x_k > x'_k$ (eine Zahl verkleinern, andere bleiben stehen, Resultat ist monoton)

  Bsp: $[1,3,6] \to [1,2,6], [1,3,6] \not\to [1,1,6], [1,3,6] \not\to [1,0,6]$

- Aufgabe: berechne Spielwerte! ergänze `ki-ss23/nim instance Game Sch where next ...`
- A: reduziere (die Bestimmung der P/N-Bewertung von) Schiebe-Nim auf Nim! Hinweis: gerade/ungerade

Gomoku-Heuristik

- Author: Phillipe Schnoebelen, Quelltext: `/usr/share/emacs/28.2/lisp/play/gomoku.el.gz` , Abschnitt „The Score Table“.
- Wert für leeres Feld p (als möglicher Zug)

  ist Summe der Werte der einfarbigen 5-Tupel q mit $p \in q$.

- Heuristik ist 9-Tupel $[m_0, \ldots, m_4, d_1, \ldots, d_4]$ von Werten

  m_k/d_k für Tupel mit k von meinen/deinen Steinen,

- Implementierung siehe `ki-ss23/gomoku`

```
gomoku-main eval 0 1 2 100 10000 1 2 10 1000
... won 27 of 100 games
```

- Aufgabe (auch autotool): bessere Heuristik finden

2 Optimierung

Motivation, Beispiel, Plan

- Bestimmung guter Parameter für die Gomoku-Heuristik
- Menge der Parameter ist $P = \mathbb{R}^9$,

  Bewertungsfunktion $v : P \to \mathbb{R}$ (z.B. Anzahl der Siege in 10 Spielen gegen die randomisierte Default-Strategie)

- Folge $p_0, p_1, \cdots \in P^\omega$ mit:

  p_0 zufällig (oder geraten), p_{i+1} = Verbesserung von p_i.

- woher bekommt man diese Verbesserug?
  - blind probieren (in der Nähe)
  - zielgerichtet probieren (in bestimmte Richtung)

Naive lokale Suche (allgemein)

- Ein-Punkt-Mutation:

  eine Komponente von p_i zufällig auswählen, zufällig ändern, ergibt p',

  $p_{i+1} = (v(p') > v(p_i)p'p)$

- Varianten: wenn mehrfach keine Verbesserung, dann ...
  - p' auch akzeptieren, wenn $v(p) = v(p')$
  - ... wenn $v(p) - \delta < v(p')$
  - ganz neu starten
- Variante (evolutionäre Suche): zu jedem Zeitpunkt nicht nur ein einzelnes p, sondern eine Menge (Population)

  Vererbung, Kreuzung, Selektion (siehe separate VL)

Naive lokale Suche (für Gomoku-Heuristik)

- Bewertungsfunktion *Anzahl der Siege* S ist anfangs (und lange) $= 0$, liefert überhaupt keine Information
- $\Rightarrow$ Information auch aus Niederlagen extrahieren!

  Anzahl Z_V der Züge bis zum Verlust. (mehr $\Rightarrow$ besser)

- zusätzliche Information auch aus Gewinn: auch die Anzahl Z_G der Züge! (weniger $\Rightarrow$ besser)
- insgesamt (summiert über mehrere Spiele)

  $v(p) = (S, -\sum Z_V, \sum Z_G)$

  und aufsteigend lexikografisch geordnet
- *Anzahl der Züge* approximieren durch *Länge* (als Zeichenkette) *der Ausgabe des autotool*

Richtungs-Suche (Gradienten-Abstieg)

- wenn die Zielfunktion sich lokal (d.h., bei kleinen Änderungen im Argument) vernünftig verhält (keine Sprünge im Resultat)

  (NB: das ist für Gomoku aber nicht der Fall! - warum?)

  dann kann man die mathematische Theorie der in diesem Sinne vernünftigen Funktionen anwenden (die Differentialrechnung)

- Anwendungsfall (jetzt) Matrix-Ungleichungen f. Terminations.-Analyse

  (später) Koeffizienten für Audio-Signal-Analyse, Koeffizienten für neuronale Netze

Anw: Matrix-Interpretationen (Bsp)

- Bezeichnungen: $Q_d := \mathbb{R}_{\geq 0}^{d \times d}$ (quadratische Matrizen),
  $E_d := \{M \mid M \in Q_d \wedge M_{1,1} \geq 1 \wedge M_{d,d} \geq 1\}$,
  $P_d := \{M \mid M \in Q_d \wedge M_{1,d} \geq 1\}$
- Bsp: gesucht sind $A, B \in E_d$ mit $(A \cdot B) - (B \cdot A) \in P_d$.
  eine Lösung: $A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$
- Anw.: Ersetzungssystem $R = \{(ab, ba)\}$ über $\Sigma = \{a, b\}$:
  für Matrix-Interpretation $i : \Sigma \to E_2 : a \mapsto A, b \mapsto B$, fortgesetzt zu $i^* : \Sigma^* \to E_2$ durch $i(w) = \Pi_{c \in w} i(c)$,
  gilt: $u \to_R v \Rightarrow i^*(u)_{1,2} \geq i^*(v)_{1,2} + 1$
  also $u \to_R^k v \Rightarrow i^*(u)_{1,2} \geq k$,
  $i^*(u)_{1,2}$ ist Schranke für Länge von R-Ableitungen von u.

Matrix-Interpretationen (Def, Aufgaben)

- Bezeichnungen: $Q_d := \mathbb{R}_{\geq 0}^{d \times d}$ (quadratische Matrizen),
  $E_d := \{M \mid M \in Q_d \wedge M_{1,1} \geq 1 \wedge M_{d,d} \geq 1\}$,
  $P_d := \{M \mid M \in Q_d \wedge M_{1,d} \geq 1\}$
- Def: $i : \Sigma \to E_d$ ist *kompatibel mit* Regelmenge R über Σ,
  wenn $\forall (l, r) \in R : i^*(l) - i^*(r) \in P_d$.

- weitere Beispiele (Ü-Aufgaben): gesucht ist jeweils eine kompatible Matrix-Interpretation für:
  - $ab \to bba$ (Hinweis: $d = 2$)
  - $aa \to aba$ (Hinweis: $d = 3$)
  - $a^2b^2 \to b^3a^3$ (H: $d = 5$ ist möglich, geht auch kleiner?)
  - $\{aa \to bc, bb \to ac, cc \to ab\}$

Matrix-Interpretationen (Lösungsverfahren)

- für $ab \to ba$ mit $d = 2$. Ansatz: $A = \begin{pmatrix} p & q \\ 0 & 1 \end{pmatrix}, B = \begin{pmatrix} r & s \\ 0 & 1 \end{pmatrix}$.
  dann $AB - BA = \begin{pmatrix} 0 & ps + q - qr - s \\ 0 & 0 \end{pmatrix}$
- Parameter: $p, r \geq 1, q, s \geq 0$, Ziel $Z = ps + q - qr - s$ (≥ 1)
- wählen Startwert $S_0 = (1, 1, 1, 1)$, dann $Z(S_0) = 0$.
- Änderung von Z bei Änderung einer Komponente von S_0:
  $\frac{\partial Z}{\partial p} = s = 1$, $\frac{\partial Z}{\partial q} = 1 - r = 0$, $\frac{\partial Z}{\partial r} = -q = -1$, $\frac{\partial Z}{\partial s} = p - 1 = 0$.
  Z erhöhen: p vergrößern (OK), r verkleinern (verboten)
  wähle $S_1 = (2, 1, 1, 1)$, $Z(S_1) = 1$, ist Lösung.

Lösungsverfahren, Diskussion

- Par.: $p \geq 1, q \geq 0, r \geq 1, s \geq 0$, Ziel $Z = ps + q - qr - s$ (≥ 1)
- Gradient $\nabla Z = (s, 1 - r, -q, p - 1)$
- für Startwert $S_0 = (1, 1, 1, 1)$ hatten wir etwas Glück.
  betrachte $S_0 = (1, 0, 1, 0)$. Dann $\nabla Z_0 = (0, 0, 0, 0)$.
  Wohin laufen? (work-around: anderes S_0 würfeln)

- im Allg. haben wir mehrere Bedingungen,

  $i^*(l) - i^*(r) = D$ mit $D_{1,1} \geq 0, D_{1,2} \geq 1, D_{2,1} \geq 0, D_{2,2} \geq 0$.

  müssen wir ausdrücken als *ein* Optimierungsziel,

  eigentlich $Z = \min(D_{1,1}, D_{1,2} - 1, D_{2,1}, D_{2,2})$ (soll ≥ 0)

  ... aber dieses Z ist nicht überall differenzierbar

autotool-Aufgabe zu Matrix-Interpretationen

- der Aufgabentyp ist für den allgemeineren Fall vorgesehen (Term-Ersetzung, statt Wort-Ersetzung), deswegen sehen die Wörter und Interpretationen anders aus.

  Wörter werden dargestellt durch Terme über eine Signatur mit einstelligen Symbolen. Die Wort-Ersetzungs-Regel $ab \to ba$ wird dargestellt als Term-Ersetzungs-Regel $a(b(x_1)) \to b(a(x_1))$.

  Die Matrix $\begin{pmatrix} p & q \\ 0 & 1 \end{pmatrix}$ wird notiert als die (einstellige) lineare Funktion $x_1 \mapsto p \cdot x_1 + q$ mit Absolutglied q und linearem Koeffizienten p.

  Die letzte Zeile unsere Matrix wird also nicht notiert (sie ist immer $0, \ldots, 0, 1$) und die rechteckige (nicht quadratische) Teilmatrix darüber wird zerlegt in eine quadratische (p) und einen Spaltenvektor (q).

- die auf vorigen Folien angegebene Interpretation ist

```
Interpretation
  { dim = 1
  , values = Matrix_Interpretation_Natural (listToFM
      [ ( b
        , Multilinear
          { absolute = column[1], coefficients = [ matrix [[1]]] }
        )
      , ( a
        , Multilinear
          { absolute = column[0], coefficients = [ matrix [[2]]] }
        )
      ])
  }
```

  das löst die Aufgabe 16-0.

- in der System-Antwort steht

  ```
  interpretation of a ( b ( 1 ) )
  ```

  u.ä., dabei bezeichnet `1` die Variable x_1, also `a(b(1))` die linke Regelseite.

Aufgaben

1. Parameter für Gomoku-Heuristik:

   - Probieren Sie lokale Suche ohne Vorwissen ($p_0 = [0, 0, 0, 0, 0, 0, 0, 0, 0]$),
   - lokale Suche zur Verbesserung einer bekannten guten Lösung.
   - Für die angegebene Bewertung mit $(S, -\sum Z_V, \sum Z_G)$: vorhersagen und ausprobieren, wie sich das Verhalten ändert, wenn Komponenten dieses Tripels vertauscht oder gelöscht werden.
   - Welches ist die beste Anzahl von Spielen je Bewertung? (mehr $\Rightarrow$ genauer, aber langsamer)
   - Gefundene Parameter in autotool eingeben oder in Issue schreiben, dann Turnier unter diesen Heuristiken.

2. Matrix-Interpretationen für Termination:

   - Geben Sie kompatible Interpretationen an für $ab \to bba$, für $aa \to aba$. (Andere Beispiele auf Folie sind schwieriger)
   - Lösen Sie $ab \to bba$ mit Ansatz $\begin{pmatrix} * & * \\ 0 & 1 \end{pmatrix}$ durch Gradienten-Abstieg von einem gewürfelten oder geratenen Startwert aus.
   - desgleichen für $aa \to aba$. Vorsicht: das geht nicht (man braucht Dimension 3). Wie sehen die Gradienten aus?

   Symbolisches Rechnen mit

   ```
   $ rlwrap maxima
   (%i1) a:matrix([p,q],[0,1]);
                                        [ p  q ]
   (%o1)                                [      ]
                                        [ 0  1 ]
   (%i2) b:matrix([r,s],[0,1]);
   ```

```
                                    [ r  s ]
(%o2)                               [      ]
                                    [ 0  1 ]
(%i3) a.b - b.a;
                          [ 0  p s - s - q r + q ]
(%o3)                     [                      ]
                          [ 0          0         ]
(%i9)  subst([p=1+d,q=1,r=1,s=1],a.b - b.a);
                                    [ 0  d ]
(%o9)                               [      ]
                                    [ 0  0 ]
```

3. Gradienten-Abstieg: bestimmen Sie ein Minimum der Funktion x^2+y^2 durch Gradienten-Abstieg von $(x = 1, y = 2)$ aus.

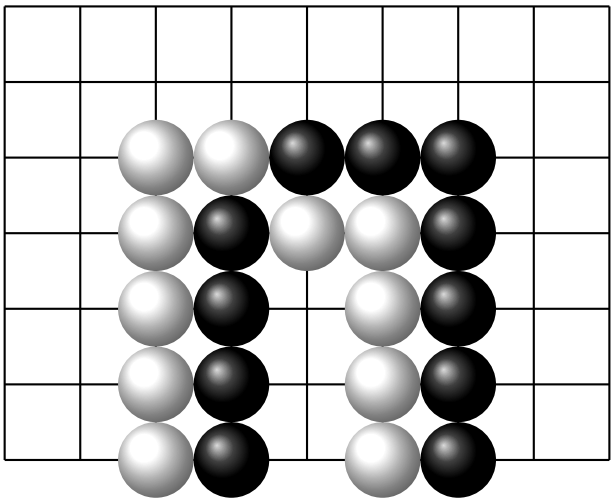
   Desgl. für $(x^2 - y^2 + 1)^2$ oder eine andere selbst gewählte oder gefundene Funktion.

   Probieren Sie jeweils verschiedene Schrittweiten.

   Funktionsgraphen (Flächen) ansehen (und drehen!) mit

```
$ gnuplot
gnuplot> splot [-2:2][-2:2] (x*x-y*y+1)**2
```

4. Damit wir das Go-Spiel nicht vergessen: unter der Annahme, daß die äußeren Gruppen leben: was ist der Status der 4 schwarzen und 5 weißen Steine innen?



3 Symbolische Differentiation

Motivation, Plan, Ausblick

- Ableitungen exakt ausrechnen (eine Grundaufgabe der Computeralgebra, Anwendung: Optimierung)
- ähnlich: Algorithmus von Brzozowski (1964) zur Synthese eines deterministischen endlichen Automaten (DFA) aus einem regulären Ausdruck durch Ableitungen (Suffix-Mengen)
- für den Gradienten-Abstieg wird dann aber einen andere Methode verwendet (*automatic differentiation*: eine Mischung aus numerischer und symbolischer Rechnung)
- trotzdem heute dieser Ausflug in die Geschichte der KI

Algebraische Repräsentation von Funktionen

- durch Terme dieser Signatur:

```
data F c v = Con c | Var v
  | Plus (F c v) (F c v) | Times (F c v) (F c v)
  | Div (F c v) (F c v)  | Exp (F c v)
```

- Bsp: `t = Plus (Times (Con 2) (Var X)) (Var Y)` repräsentiert $2 \cdot x + y$
- eine Funktion an einer Stelle auswerten:

  ```
  eval :: M.Map v c -> F c v -> c
  ```

  Bsp: `eval (M.fromList [(X,2),(Y,0)]) t`

Tricks für kurze Quelltexte

- `Plus (Times (Con 2) (Var X)) (Var Y)`
  repräsentiert $2 \cdot x + y$, aber sieht nicht so aus.
- im Quelltext würden wir gern schreiben `2 * x + y`
  können wir auch! nach diesen Deklarationen:

  ```
  instance Num c => Num (F c v) where
    fromInteger = Con . fromInteger ; negate x = _
    (+) = Plus;  (*) = Times
  data V = X | Y  ; x = Var X ; y = Var Y
  ```

- damit funktioniert auch `(x^2 - y^2 + 1)^2`,
  weil die Implementierung der Potenz polymorph ist:

  ```
  (^) :: (Num a, Integral b) => a -> b -> a
  ```

Symbolisches Differenzieren

- `diff :: v -> F c v -> F c v`
- Bsp: erwartete Resultate für

  `t = Plus (Times (Con 2) (Var X)) (Var Y)`,

  `diff X t = Con 2`; `diff Y t = _`
- Implementierung:

  ```
  diff v f = case f of
    Con _ -> Con 0
    Var u -> if u == v then Con 1 else Con 0
  ```

- ergänze Code für Plus, Times, Div, Exp
- Test: Vergleich von Differenzenquotient und Differentialquotient (benutze `eval`)

Anwendung: Gradienten-Abstieg

- ```
  gradient :: _ => F c v -> M.Map v (F c v)
  gradient f = M.fromList $ do
    v <- S.toList $ variables f
    return (v, diff v f)
  step :: _ => F c v -> c -> M.Map v c -> M.Map v c
  step f s b =
    let g = fmap (eval b) $ gradient f
    in  M.intersectionWith (\ x y -> x - s * y) b g
  steps f s b
    = map (\b -> (b,eval b f)) $ iterate (step f s) b
  ```

- Test: `steps f1 0.1 (M.fromList [(X,1),(Y,2)])`

  ```
  [(fromList [(X,1.0),(Y,2.0)],4.0)
  ,(fromList [(X,1.8),(Y,0.4)],16.6464)
  ,(fromList [(X,-1.1376),(Y,1.0528)],1.4059933867966474)
  ```

Äquivalenzen, Normalformen

- die `data`-Deklaration definiert eine freie Algebra,
- es gibt aber syntaktisch unterschiedliche Terme, die semantisch gleich sind: Operationen $+, \cdot$ sind assoziativ, kommutativ, besitzen neutrales Element, Distributivität
- U: Umformungs-Regeln, z.B.
  - `Plus (C 0) x -> x`
  - `Plus (Plus x y) z -> Plus x (Plus y z)`
  - `Plus (C a) (C b) -> C (a+b)`
  - `Plus x y -> Plus y x`
  - $\exp(x + y) = \exp(x) \cdot \exp(y)$
- Korrektheit, Termination, Ableitungslängen von $\rightarrow_U$?
  ```

Kanonische Repräsentationen

- anstatt nachträglicher Normalisierung:

  von Anfang an normalisierte Repräsentation.

- anstatt zweistelligem Symbol `data T = ... | G T T`
  - falls assoziativ: verwende Liste `G [T]`
  - falls assoziativ und kommutativ: Menge `G (S.Set T)`
  - Invariante: für jedes `G s` und $x \in s$: Wurzel von x ist $\neq$ `G`
  - falls neutrales Element e: Invariante: $e \notin s$
- ersetze altes `G :: T -> T -> T` durch *smart constructor* mit gleichem Typ `g :: T -> T -> T`
- mit Distributivität von Plus über Mal: verwende Polynome

  Inv: kein Kind von Mal ist Plus

Sharing, DAGs, Schaltkreise

- wiederholte Auswertung von `diff x t` für Teilterme `t`
- Lösung: Cache (Memoisierung)

  damit wird Ableitung als DAG realisiert, das kann man von Anfang an auch für die Funktion selbst so einrichten.
- eine konkrete Realisierung eines solchen DAG ist ein *Schaltkreis*
- (später) neuronale Netze *sind* solche Schaltkreise,

  Ausgang eines Neurons in Schicht k mglw. verbunden mit Eingängen mehrerer Neuronen in Schicht $> k$.

Aufgaben

1. probieren Sie weitere Normalisierungs-Regeln. Testen Sie an Beispielen, ob Ableitungen von Funktionen damit vernünftig aussehen.
2. Wenden Sie das Verfahren des Gradiententabstiegs an zum Finden von Matrix-Interpretationen.

- eine reelle Unbekannte mit einseitig beschränktem Bereich (z.B. $x \geq 1$) kann dargestellt werden als eine Funktion einer unbeschränkten reellen Unbekannten ($x = 1 + y^2$).
- eine Bedingung $A \geq B$ (für Ausdrücke A, B) kann mit einer neuen unbeschränkten Unbekannten h dargestellt werden als Gleichung $A = B + h^2$.
- die Erfüllung eines Gleichungssystems $A_i = B_i$ kann als Minimierungsaufgabe für $\sum_i (A_i - B_i)^2$ dargestellt werden.

Durch die angegebenen Umformungen steigen die Anzahl der Unbekannten und der Grad der Polynome. Beides erschwert die Arbeit des Verfahrens.

Probieren Sie zunächst, nach diesem Verfahren ein einfaches lineares Gleichungssystem zu lösen ($2x + 3y = 1, x - y = 1$), dann ein lineares Ungleichungssystem, dann eine quadratische Gleichung ($x^2 + x = 1$).

4 Differentiation regulärer Ausdrücke

Motivation

- Ableitung eines Polynoms: $\frac{\partial}{\partial x} x^3 = 3x^2$
- Links-Quotient einer Sprache: $L/x = \{w \mid x \cdot w \in L\}$,

  Berechnung: Differentiation von regulären Ausdrücken

  Bsp: $\frac{\partial}{\partial a}\{aaa\} = \{aa\}$, $\frac{\partial}{\partial a}\{ba\} = \emptyset$, $\frac{\partial}{\partial a} a^* = a^*$, $\frac{\partial}{\partial a} a^* b^* = ?$,
- wiederholte Ableitung eines regulären Ausdrucks X nach allen Buchstaben ergibt deterministischen (mglw. unendlichen) Automaten A mit $\mathsf{Lang}(A) = \mathsf{Lang}(X)$.

  durch geeignete Ableitungs- und Normalisierungs-Regeln

  erreicht man Endlichkeit (J. Brzozowski, 1964)

Reguläre Ausdrücke

- ```
  data RX c = Empty | Epsilon | Letter c
    | Union (RX c) (RX c) | Dot (RX c) (RX c)
    | Star (RX c)
  ```
- `Union`: assoziativ, kommutativ, Einheit `Empty`

  `Dot`: assoziativ, nicht kommutativ, Einheit `Epsilon`
  ```

- Ableitung von Ausdrücken (Typ und Spezifikation) `diff :: c -> RX c -> RX c`; $\mathsf{Lang}(\texttt{diff}(c, X)) = \mathsf{Lang}(X)/c = \{w \mid c \cdot w \in \mathsf{Lang}(X)\}$.
- Implementierung (Anfang)

```
diff c e = case e of
  Empty -> Empty ; Epsilon -> _
  Letter c' -> if c == c' then _  else _
  Union l r -> Union _ _
```

Ableitung eines Produktes

- einfacher Fall (Ableitung des linken Faktors reicht aus)
  $\frac{\partial}{\partial a}(a \cdot \{b, c\}) = \{b, c\}$,
- Ableitung des rechten Faktors wird auch benötigt:
  $\frac{\partial}{\partial a}(\{aaa, \epsilon\} \cdot \{b, ac\}) = \{aa\} \cdot \{b, ac\} \cup \{c\}$.
- $\frac{\partial}{\partial a}(X_1 \cdot X_2) = \ldots$
  - falls $\epsilon \notin (X_1)$, dann $\frac{\partial}{\partial a}(X_1) \cdot X_2$
  - falls $\epsilon \in (X_1)$, dann $\frac{\partial}{\partial a}(X_1) \cdot X_2 \cup \frac{\partial}{\partial a}(X_2)$
- Spezifikation $N(X) \iff \epsilon \in \mathsf{Lang}(X)$, Implementierung:
  $N(\emptyset) = 0, N(\epsilon) = 1, \forall c \in \Sigma : N(c) = 0,$
  $N(X + Y) = N(X) \vee N(Y), N(X \cdot Y) = ?, N(X^*) = ?$

Ableitung eines Sterns

- es gelten $L^* = \{\epsilon\} \cup L \cdot L^*$ und $L^* = (L \setminus \{\epsilon\})^*$,
  also $L^* = \{\epsilon\} \cup (L \setminus \{\epsilon\}) \cdot L^*$,
- $\frac{\partial}{\partial a}(L^*) = \frac{\partial}{\partial a}(\{\epsilon\} \cup (L \setminus \{\epsilon\}) \cdot L^*) = \frac{\partial}{\partial a}\{\epsilon\} \cup \frac{\partial}{\partial a}((L \setminus \{\epsilon\}) \cdot L^* = \emptyset \cup \left(\frac{\partial}{\partial a}L\right) L^* = \left(\frac{\partial}{\partial a}L\right) L^*$
- Bsp $\frac{\partial}{\partial a}(aa)^* = \left(\frac{\partial}{\partial a}aa\right) \cdot (aa)^* = a \cdot (aa)^*$

Ableitung nach einem Wort

- Def: für $u = u_1 \cdots u_n$ mit $u_i \in \Sigma$: $\frac{\partial}{\partial u} L = \frac{\partial}{\partial u_n} \cdots \frac{\partial}{\partial u_1} L$.

  Satz: $\frac{\partial}{\partial u} X = \mathsf{Lang}(X)/u = \{v \mid u \cdot v \in \mathsf{Lang}(X)\}$

- Def: die Nerode-Kongruenz von L ist die Relation $u \sim_L u' \iff L/u = L/u'$

  Satz: L regulär $\iff \Sigma^*/\sim_L$ ist endlich

- Folgerung: jeder reguläre Ausdruck X hat nur endlich viele nicht äquivalente Ableitungen $\frac{\partial}{\partial u} X$ (und diese bilden den minimalen DFA für $\mathsf{Lang}(X)$)

- Beobachtung: mit unserer Implementierung `diff` entstehen sehr oft doch unendlich viele Ableitungen. (semantisch äquivalent, aber syntaktisch verschieden)

Algorithmus von Brzozowski

- besteht aus:
  - Ableitungsregeln (`diff`)
  - Normalisierungs-Regeln

  so daß jeder reguläre Ausdruck nur endlich viele syntaktisch verschiedene Wort-Ableitungen hat

- Normalisierungs-Regeln realisieren für `Union`: Assoziativität, Kommutativität, Neutralität von `Empty`.

- alternative Implementierung: ersetze `Union (RX c) (RX c)` durch normalisierte Repräsentation `Unions (S.Set (RX c))`

Aufgaben

1. Untersuchen Sie (an Beispielen), ob der von unserer Implementierung des Algorithmus von Brzozowski berechnete DFA minimal ist.

2. Diskutieren Sie Normalisierungs-Regeln und normalisierte Repräsentation für `Dot`.

5 Neuronale Netze

Zusammenfassung

- ein neuronales Netz N ist ein DAG, der eine Funktion $S_N : \mathbb{R}^{\text{Parameter}} \times \mathbb{R}^{\text{Eingaben}} \to \mathbb{R}^{\text{Ausgaben}}$ realisiert

  $x \mapsto S_N(P, x)$ soll durch gegebene Stützstellen verlaufen
- typische Anwendung: Klassifikation ($|\text{Ausgaben}| = 1$),
  Stützstellen sind $\subseteq \{(x, 1) \mid x \in C\} \cup \{(x, 0) \mid x \notin C\}$
- Struktur von N passend zu Aufgabenstellung
  - Konvolutions-Schicht — Translations-Invarianz
  - kleine Zwischenschichten erzwingen Kompression
- Bestimmen d. Parameterwerte durch Gradienten-Abstieg

Geschichte, Quellen

- Warren McCulloch, Walter Pitts: *A Logical Calculus of Ideas Immanent in Nervous Activity*, 1943, `https://doi.org/10.1007/BF02478259`
- Steven C. Kleene: *Representation of Events in Nerve Nets and Finite Automata*, in: Claude Shannon, John McCarthy: *Automata Studies*, 1956, `https://doi.org/10.1515/9781400882618`
- Yann Le Cun, Yoshua Bengio: *Word-level training of a handwritten word recognizer based on convolutional neural networks*, ICPR 1994, `https://doi.org/10.1109/ICPR.1994.576881`

Definition

- Syntax: ein neuronales Netz N
  ist ein gerichteter, geordneter, kreisfreier Graph G
  mit Knotenmenge: $V = \text{Eingabe} \cup \text{Parameter} \cup \text{Neuronen}$,
  enthält (Folge von) Ausgabeknoten $A \in V^*$
  jedem Neuron n ist Fkt. $F(n) : \mathbb{R}^{\mathsf{indeg}(n)} \to \mathbb{R}$ zugeordnet.

- Bsp: Eingaben: $E = \{e_1, e_2\}$, Parameter $P = \{p_0, p_1, p_2\}$,
  Neuron $n_1 : p_0 + p_1 \cdot e_1 + p_2 \cdot e_2$,
  Neuron $n_2 : \max(0, \min(1, n_1))$, Ausgabe $A = [n_2]$
- Semantik: ist Funktion $S_N : \mathbb{R}^{|P|} \times \mathbb{R}^{|E|} \to \mathbb{R}^{|A|}$
  Bewertung $b : V \to \mathbb{R}$ entlang einer topologischen Ordnung des DAG, für Neuron n mit Funktion f und Eingängen $v_1, \ldots, v_k$ gilt $b(n) = f(b(v_1), \ldots, b(v_k)$

Modellierung von NN

- ```
  data Function arg
    = Dot_Product [arg] [arg] | Normalize arg
  data Ref e = Intern Index | Extern e
  data Net e = Net -- Syntax
    { graph :: M.Map Index (Function (Ref e) }
  data Source = Param Int | Input Int
  n0 :: Net Source
  n0 = Net { graph = M.fromList
    [ (1, Dot_Product [ Extern (Param 1), ..
  ```
- ```
  evaluate  -- Semantik
    :: Num z => (e -> z) -> Net e -> M.Map Index z
  ```
- vollst. Quelltext siehe Repo zur VL

Spezielle Formen neuronaler Netze

- spezielle Funktionen:
  - Linearkombinationen: $f : \vec{x} \mapsto p_0 + \sum p_i \vec{x}_i$
  - Normalisierung auf geschlossenes Intervall $[0, 1]$
    - $x \mapsto \max(0, \min(1, x))$ stetig, monoton
    - $x \mapsto 1/(1 + \exp(-x))$ stetig, streng monoton, diff-bar
- spezielle Graphen:
  - flache Netze: Eingabe-Schicht, Ausgabe-Schicht
  - nicht flache („deep“) Netze: weitere („versteckte“) S.
- mehrfache Benutzung von Parametern: Konvolutionen

- diese (und andere) Spezialformen motiviert durch:
  - Nachbildung natürlicher Neuronen
  - Anwendg.sbezug (2D-Muster $\rightarrow$ Translationsinvarianz)

Parameterbestimmung für NN

- zu lösen ist diese (Optimierungs-)Aufgabe:
  - gegeben: Netz N, Stützstellen $T \subseteq E \times A$
  - gesucht: Parameter p, so daß
    - (exakte Aufgabe) $\forall (e, a) \in T : a = S_N(p, e)$
    - (Optimierung) $\sum_{(e,a)\in T}(a - S_N(p, e))^2$ minimal
- iteratives Verfahren: p_0 zufällig, dann für $i = 0, 1, \ldots$
  - (e, a) zufällig aus T,
  - Gradient $G = \nabla(p \mapsto (a - S_N(p, e))^2)$, ist Fkt: $\mathbb{R}^P \rightarrow \mathbb{R}$
  - $p_{i+1} = p_i - c \cdot G(p_i)$
- Satz: wenn ..., dann exist. $\lim p_i$ und ist *lokales* Min.
- Anwendung: wir wollen *globales* Min. unter realen Bed.

Verfahren zur Gradientenbestimmung

- löst diese Aufgabe:
  - gegeben: Vektor $x \in \mathbb{R}^d$, arithmetische Fkt. $f : \mathbb{R}^d \rightarrow \mathbb{R}$ als DAG (z.B. NN, Datenflußgraph eines Programms)
  - gesucht: $(\nabla f)(x)$, Wert des Gradienten an Stelle x
- Lösungsverfahren:
  - *symbolische* Differentiation: Term für ∇f
    ... der kann aber sehr groß werden
  - *numerische* Diff.: berechnet $(f(x + h \cdot \vec{e_i}) - f(x))/h$
    ... anfällig für Rundungs- und Auslöschungs-Fehler

  - *automatische* Diff.: symbolische Rechng. auf DAG von f

- Baydin, Pearlmutter, Radul, Siskind: *Automatic differentiation in machine learning: a survey*, 2015–2018 `https://arxiv.org/abs/1502.05767`

Automatische Differentiation (AD)

- für DAG, die arithm. Fkt. repräsentieren:
- für jeden Knoten nicht nur Funktionswert,

  sondern auch Wert des Gradienten bestimmen
- repräsentiert durch „Zahlen“ der Form

```
data N v = N { absolute :: Double
             , linear :: M.Map v Double }
```

- dafür die arithm. Op. implementieren

```
instance Ord v => Num (N v) where
  x + y = _  ; x * y = _ ; exp x = _
```

- `evaluate :: Num w => (e -> w) -> Net e -> ...`

  kann für `w = N v` unverändert benutzt werden!

Stochastischer Gradientenabstieg

- Bsp: gesucht ist Netz mit zwei Schichten für XOR.
- volles Optimierungsproblem: minimiere Fehlerquadratsumme über alle (4) Eingabe/Ausgabe-Paare (benutze Gradient dieser Fkt.)
- Variante: in jedem Schritt diese Summe über ein zufällige kleine Teilmenge (Bsp: 2) von E/A-Paaren
  - weniger Rechenaufwand pro Schritt
  - Ausbruch aus lokalen Minima (voller Gradient $= 0$), die global schlecht sind
- Variante: je Schritt: betrachte eine zufällige Teilmenge der Parameter als konstant (Gradient nur für die anderen)
  - weniger Rechenaufwand pro Schritt

Diskussion (Beispiel)

- für unser Beispiel (XOR) ist NN sinnlos, da man
  - XOR(x, y) direkt exakt und schnell ausrechnen kann
  - ...es mehr Parameter (9) als Stützstellen (4) gibt
- Aufwand für Parameter-Optimierung lohnt sich nur, wenn |Testfälle| $\ll$ |Anwendungsfälle|
- aber dann ist fraglich, ob das Netz die Eingaben $e \in$ Anwendungsfälle $\setminus$ Testfälle richtig behandelt.

Diskussion (allgemein)

Gary Marcus: *Deep Learning: A Critical Appraisal*, 2018, `https://arxiv.org/abs/1801.00631`

The real problem lies in misunderstanding what deep learning is, and is not, good for. The technique excels

- at solving *closed-end classification problems*,
- in which a *wide range of potential signals*
- must be mapped onto a *limited number of categories*,
- given that there is *enough data available*
- and the test set *closely resembles the training set*.

Deviations from these assumptions can cause problems. Deep learning is just a statistical technique. All statistical techniques suffer from deviation from their assumptions.

Aufgaben

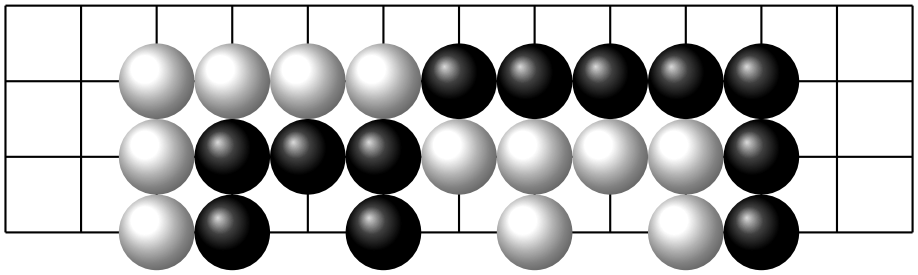
1. zu den Quelltexten zu NN und AD: Parameterbestimmung für Repräsentation unterschiedlicher Boolesche Funktionen verschiedener Stelligkeit durch unterschiedlicher Netz-Topologie und unterschiedlichen Optimierungsverfahren: ausprobieren und Beobachtungen diskutieren.
2. speziell: welche Form paßt zu dreistelligem XOR? dreistelligem *exactly 2*? vierstellig?

3. implementieren Sie AD für die Normalisierung durch $x \mapsto \max(0, \min(x, 1))$, vergleichen Sie dann das Verhalten (in o.g. Anwendungen) mit Normalisierung durch $x \mapsto 1/(1 + \exp(-x))$.

4. die zweite Variante der stochastischen Einschränkung implementieren (in jedem Schritt einige Parameter fixieren)

5. für die Funktion $g : x \mapsto 1/(1 + \exp(-x))$: beweisen Sie, daß g durch 0 und 1 beschränkt ist, streng monoton steigt, durch $(0, 1/2)$ verläuft und in diesem Punkt drehsymmetrisch ist.

6. unter der Annahme, daß die äußeren Gruppen leben: was ist der Status innen?



6 Neuronale Netze: (kleine) Beispiele

Motivation

- welche Funktionen $S_N : \mathbb{R}^{\text{Parameter}} \times \mathbb{R}^{\text{Eingaben}} \to \mathbb{R}^{\text{Ausgaben}}$ lassen sich durch Netze realisieren?
- falls realisierbar: werden die Parameter durch (stoch.) Gradienten-Abstieg tatsächlich (schnell) gefunden?
- die Antworten hängen ab von:
  - Typ der Neuronen (hier: Linearkombination mit Verschiebung, mit sofort folgender Normalisierung)
  - Anzahl der Neuronen (hier: sehr wenige)
  - Verbindungen der Neuronen (hier: beliebig)

Ein Neuron

- n Eingänge $x_1, \ldots, x_n$, $(n+1)$ Parameter $p_0, p_1, \ldots, p_n$,
  berechnet $\mathsf{Norm}(p_0 + \sum p_i \cdot x_i)$
  Linearkombination mit Verschiebung (p_0), Resultat normalisiert durch: $\mathsf{Norm}(x) = 1/(1+\exp(-x))$.
- Darstellung logischer Fkt. $\mathbb{B}^n \to \mathbb{B}$ mit $\mathbb{B} = \{0, 1\}$:
  - Nicht exakt (wg. Norm), aber beliebig genau möglich: Id, Not, And, Or, Atleast(k), Atmost(k).
  - Darstellbare Fkt. $f : \mathbb{R}^n \to \mathbb{R}$ sind in jedem Parameter monoton oder konstant: für jedes $1 \leq i \leq n$ gilt entweder $\forall x_i, x_i' : x_i < x_i' \Rightarrow f(\ldots, x_i, \ldots) < f(\ldots, x_i', \ldots)$ oder $\forall x_i, x_i' : x_i < x_i' \Rightarrow f(\ldots, x_i, \ldots) > f(\ldots, x_i', \ldots)$ oder $\forall x_i, x_i' : f(\ldots, x_i, \ldots) = f(\ldots, x_i', \ldots)$
    Aber $\mathsf{Xor}(\underline{0}, 0) < \mathsf{Xor}(\underline{1}, 0) = \mathsf{Xor}(\underline{0}, 1) > \mathsf{Xor}(\underline{1}, 1)$

Xor mit zwei Neuronen

- Basisfunktionen (Not, And, Or) sind darstellbar

  $\Rightarrow$ jede Fkt $\mathbb{B}^n \to \mathbb{B}$ ist als Schaltkreis darstellbar.

  (Zu zeigen ist: ... beliebig genau darstellbar)
- also $\mathsf{Xor}(a, b) = (\neg a \wedge b) \vee (a \wedge \neg b)$,

  mit drei Neuronen (nicht 5, denn Negation ist kostenlos)
- Darstellung mit nur zwei Neuronen:

  $n_1 = \mathsf{Norm}(4 - 10x - 10y)$, $n_2 = \mathsf{Norm}(16 - 10x - 10y - 22n_1)$

  Ü: aussagenlogische Interpretation?

  Ü: kleinste beliebig genaue Darstellung des Xor auf 3 Eingängen, ...

4-stelliges Xor

- beliebig genaue Darstellung von Xor_4 mit zwei Neuronen scheint nicht möglich
- Gradienten-Abstieg konvergiert häufig gegen eine Darstellung mit nur einem Fehler, Bsp: $f(1,1,1,1) = 1$.

- das geschieht ebenso häufig für Netz mit 3 Neuronen—obwohl das Xor_4 repräsentieren kann
- um die Parameter öfter/sicherer zu finden:

  benutze Zielfunktion $\sum_{i\in\mathbb{B}^4}(\mathsf{Xor}(i) - \mathsf{Netz}(p,i))^4$.

  dadurch werden große Fehler stärker bestraft

  Optimierung findet schnell Plateau mit $\mathsf{Netz}(p,i) \approx 0.5$, das wird sehr vorsichtig verlassen

Darstellungen einstelliger Funktionen

- ein Neuron: Fkt. ist monoton, antiton oder konstant.
- mehrere Neuronen: Fkt. mit mehreren Wendepunkten sind darstellbar
- Bsp: für $f : 1 \mapsto 1, 2 \mapsto 0, 3 \mapsto 1, 4 \mapsto 0$

  $n_1 = \mathsf{Norm}(-17 + 7x)$, $n_2 = \mathsf{Norm}(14 - 9x + 20n_1)$

  Erklärung?
- Ü: einstelliges Netz mit k Neuronen, dessen dargestellte Fkt. (deutlich) mehr als k Wendepunkte hat

Mustererkennung durch Faltung

- Bsp: die Bitfolgen aus $\mathbb{B}^5$, die das Teilwort 11 enthalten
- ein Netz dafür (mit Fehler < 0.1) ist $n_1 = \mathsf{Norm}(6.3 + 1.9x_1 + 4.8x_2 - 0.8x_3 - 6.9x_4 - 3.6x_5)$, $n_2 = \mathsf{Norm}(2.3 + 5.3x_1 + 10.6x_2 + 5.2x_3 + 0.2x_4 + 0.2x_5 - 15.7n_1)$

  aber das nicht systematisch und nicht verallgemeinerbar
- die Eigenschaft („enthält 11“) ist translations-invariant $\Rightarrow$ sollte das Netz (wenigstens eine Schicht) auch sein
- Faltungs-Neuronen n_i mit Eing. x_i, x_{i+1} (Translation)

  sowie Resultat-Neuron n_r (Eingänge $n_1, \ldots, n_4$)

  die Faltungs-Neuronen haben identische Parameter!

  Parameter insgesamt: 3 (für $n_1, \ldots, n_4$) $+ 5$ (für n_r)

Rekurrente Netze

- Bsp (Motivation) $f : \mathbb{B}^n \to \mathbb{B}^n : x \mapsto (x + 1) \pmod{2^n}$
- Berechnungsmodell ist: `mapAccumL`

```
:: (s -> a -> (s, b)) -> s -> [a] -> (s, [b])
```

- Anwendung für Nachfolger (`c`: Übertrag, `d`: Ziffer)

```
mapAccumL (\ c d -> (c && d, xor c d)) True
        [True,True,False,True]
==> (False,[False,False,True,True])
```

- im Netz für `mapAccumL f s0 xs` wird f realisiert als gemeinsame Parameter für Neuronen für jede Anwendung von f
  zwei reichen aber im Bsp. nicht (wg. Xor)
- einfacheres Beispiel: $f[x_1, \ldots, x_n] = [0, \neg x_1, \ldots, \neg x_{n-1}]$

Aufgaben

1. ist die (zweistellige) Implikation durch ein Neuron beliebig genau darstellbar? Falls ja: geben Sie für jedes $\epsilon > 0$ Parameter an, so daß die Ausgabe für alle (4) Eingaben einen absoluten Fehler $< \epsilon$ hat.

2. Falls (abweichend von unserer Darstellung) für die Normalisierung die Funktion $x \mapsto \max(0, x)$ verwendet wird: zeigen Sie, daß die auf der Folie *Ein Neuron* genannte Monotonie-Eigenschaft nicht gilt. Das Xor_2 ist trotzdem nicht durch ein solches Neuron darstellbar. Beweisen Sie das durch eine geeignete Monotonie-Aussage.

3. effiziente Darstellungen von Xor_k für $k \geq 4$

4. Geben Sie ein gute Darstellung der einstelligen Funktion $x \mapsto x^2$ auf dem Intervall $[0, 1]$ an. Welchen Absolutfehler erreichen Sie bei vollem Netz mit $1, 2, 3$ Neuronen? Verwenden Sie als Zielwerte $2, 3, 5, 9, \ldots$ gleichmäßige Stützstellen

5. folgendes Netz erkennt die Menge der 5-stelligen Binärzahlen, die durch 3 teilbar sind, mit Fehler < 0.1.

   $n_1 = \mathsf{Norm}(5 - 6x_1 + 6x_2 - 6x_3 + 6x_4 - 6x_5)$, $n_2 = \mathsf{Norm}(-13 + 5x_1 - 5x_2 + 5x_3 - 5x_4 + 5x_5 + 16n_1)$.

   Warum funktioniert das? (ein Beispiel, allgemein) Verallgemeinerung auf mehr Stellen? auf Teilbarkeit durch 5?

6. das rekurrente Netz für den binären Nachfolger bauen

Programmatische Beschreibung von Netzen

- bisher: explizite globale Numerierung der Neuronen
- alternativ: implizite globale Numerierung

```
nrecure ai = OS.net $ do
  xs <- OS.inputs ai
  let neu w = do
        ps <- OS.params $ 1 + w
        return $ \ xs -> OS.ndp ps $ One : xs
  f <- neu 2; g <- neu 3
  let step c d = let x = f [c,d]; y = g [c,d,x] in (x, y)
  s <- OS.param
  return $ snd $ L.mapAccumL step s xs
```

Verbesserter Gradientenabstieg

- Diederik P. Kingma, Jimmy Lei Ba: *Adam: A method for stochastic optimization*, ICLR 2015,

  `https://arxiv.org/abs/1412.6980`

  an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters.

7 Große (Tiefe) Neuronale Netze

Motivation, Plan, Quelle

- für große Eingaben (Bsp: alle Pixel eines Bildes, Werte eines Audio-Signals, Zeichen/Silben eines Textes)
- volles Netz (Neuron $k+1$ sieht alle Eingaben sowie Ausg. der Neuronen $1 \ldots k$): viele Parameter, große Tiefe

- geschichtete Faltungsnetze:

  wenige Parameter (diese mehrfach verwendet), flach

- Francois Fleuret: *The Little Book of Deep Learning*, 2023, `https://fleuret.org/public/lbdl.pdf` , II: Deep Models

- Anwendungen: Mustererkennung (Verkehrszeichen, Schrift), Zugvorschläge und Stellungsbewertung in Go.

Tensoren

- Idee: Tensor = mehrdimensionaler Block

- Def: Tensor T der Abmessung (Dimension) $D_1 \times \ldots \times D_k$ mit $D_i \in \mathbb{N}$ ist Abb. $T : [0 \ldots D_1 - 1] \times \ldots \times [0 \ldots D_k - 1] \to \mathbb{R}$

  Bsp: Zahl ($k = 0$), Vektor (1), Matrix (2), Quader (3), ...

- Anwendungen:

  Tensor: $1000 \times 1000 \times 3$: Bild mit 3 Farbkanälen.

  Tensor: $19 \times 19 \times 2 \times 5$: Go-Brett ($19 \times 19$)

  mit Steinen jeder Farbe (2) und den letzten 5 Zügen

Tensor-Operationen

- Addition: komponentenweise, für zwei Tensoren mit identischen Abmessungen

- Multiplikation: für zwei Tensoren mit mglw. unterschiedlichen Abmessungen—mit *einer* übereinstimmenden Komponente

  Bsp: $S : 2 \times \boxed{3} \times 5, T : \boxed{3} \times 8$,

  $S \times_{2,1} T : 2 \times 5 \times 8$

  der Index $2, 1$ bezeichnet Positionen der Komponente (in S, in T), bzgl. derer multipliziert wird

  $(S \times_{2,1} T)(a, c, q) := \sum_{b=p} S(a, b, c) \cdot T(p, q)$

  ist Verallgemeinerung des Skalarproduktes

Die voll verbundende Schicht

- realisiert Tensorprodukt mit Verschiebung (Addition)
- besteht aus Matrix $W : D \times D'$ und Vektor $b : D'$
- erzeugt aus Vektor $v : D$ einen Vektor $v' : D'$
  durch $v \mapsto v \cdot W + b$
- jeweils die Spalte i von W und die Komponente i von b
  entsprechen einem Neuron (nur Skalarprodukt, ohne Normalisierung)
  d.h., Schicht ist D' Neuronen mit je D Eingängen
- (verallgemeinert) aus Tensor: $D_1 \times \ldots \times D_k \times D$ einen Tensor: $D_1 \times \ldots \times D_k \times D'$

Die Faltungs-(Konvolutions-)Schicht

- 1-dimensional:
  - eine voll Schicht $K \to 1$ (ein Neuron)
    wird angewendet auf jeden K-Abschnitt eines Vektors $v : T$ mit $T \geq K$,
    es entsteht Vektor: $(T - K + 1)$
  - verallgemeinert: Schicht $K \to K'$, ensteht $(T - K + 1) \times K'$
  - (verallgemeinert) auf Tensor: $D_1 \times \ldots \times D_k \times T$,
    entsteht Tensor: $D_1 \times \ldots \times D_k \times (T - K + 1) \times K'$,
- 2-dimensional: Abbildung $K_1 \times K_2 \to K'$
  angew. auf Tensor: $D_1 \times \ldots \times D_k \times T_1 \times T_2$ mit $T_i \geq K_i$
  entsteht $D_1 \times \ldots \times D_k \times (T_1 - K_1 + 1) \times (T_2 - K_2 + 1) \times K'$

Anwendung: einfache Mustererkennung

- ein binäres 2D-Bild soll klassifiziert werden: ist horizontaler Block, vertikale Block, isolierter Pixel
- gesucht ist Abb. $B \times H \to 3$
- Realisierung: für jede Klasse eine 2D-Faltungsschicht $3 \times 3 \to 1$, danach eine Schicht (ein Neuron) zur Bestimmung des Maximums

- `cabal run net-main -- Adamax Img 16 Conv2`
  $\begin{pmatrix} 12 & 12 & 12 \\ -2 & -2 & -2 \\ -2 & -2 & -2 \end{pmatrix} \cdot v, \begin{pmatrix} -4 & 12 & -4 \\ -4 & 12 & -4 \\ -4 & 12 & -4 \end{pmatrix} \cdot v, \begin{pmatrix} 0 & -24 & 0 \\ -24 & 24 & -24 \\ 0 & -24 & 0 \end{pmatrix} \cdot v$
- Übung: diagonale Linien erkennen, Quadrate, Kreise
- Übung: Quelltexte verbessern (Tensor-Op. deutlicher)

Anwendung: AlphaGo Zero

- David Silver et al., *Mastering the Game of Go without Human Knowledge*, Nature, 2017.

  Abschnitt *Neural Network Architecture*

- Übung: aus dem Quelltext oder Beschreibung von KataGo die Netzwerk-Architektur herausfinden.

  David J. Wu, *Accelerating Self-Play Learning in Go*, 2020. `https://arxiv.org/abs/1902.10565`

Aufgaben

1. zu Adam/Adamax: begründen Sie anhand des Pseudocodes aus der Quelle (Abschnitt 7.1, Algorithm 2) diese Behauptungen (aus 1. Introduction)

   - the method computes individual adaptive learning rates for different parameters
   - the magnitudes of parameter updates are invariant to rescaling of the gradient
   - it naturally performs a form of step size annealing.

   Ändern Sie die Parameter α, β_1, β_2 im Quelltext, beobachten und diskutieren Sie die Auswirkungen

2. Beschreiben Sie die folgende Verarbeitung durch Tensor-Operationen. Geben Sie für jede Operation (Schicht) die Anzahl der Paramater an sowie die Abmessungen des resultierenden Tensors.

   - es liegen Meßdaten für ein Volumen $30 \times 20 \times 10$ vor (z.B. Luftdruck oder Luftfeuchte in verschiedener geografischer Position und Höhe)
   - 3D-Faltung mit Kernel $(3 \times 3 \times 3)$
   - eine voll verbundene Schicht reduziert auf zwei Dimensionen (Projektion entlang der Höhe)
   - 2D-Faltung mit Kernel 5×4

   Geben Sie für einen Ausgabe-Wert der letzten Schicht an, von welchen originalen Eingabe-Daten und welchen Parametern er abhängt.

3. durch eine 1D-Faltung mit Kernel K entsteht aus Vektor $v : T$ ein Vektor $v' : T - K + 1$.

   Geben Sie Matrix und Verschiebungsvektor einer dazu äquivalenten voll verbundene Schicht an.

   (konkretes Beispiel: $T = 4, K = 2$)

4. Für eine der Abbildungen aus Fleuret: LBDL: geben Sie die exakten Abmessungen der Tensoren an (mit konkreten Zahlen)

5. Für AlphaGo Zero, Katago: geben Sie die exakte Netzwerkstruktur an. Verwenden Sie Beschreibung oder Quelltext.

6. (Erweiterung von: *Anwendung: einfache Mustererkennung*, siehe Quelltext im Repo) Es sollen volle horizontale Linien erkannt werden. Die Trainingsdaten bestehen aus vollen horizontalen und vertikalen Linien, denen möglicherweise auch ein Pixel fehlt. Benutzen Sie ein Netz mit mehreren Faltungsschichten mit Kernel 3×3 sowie Maximum über die letzte Schicht. Wieviele Schichten sind nötig? Können alle Schichten (alle Kernel) die gleichen Parameter benutzen?

8 Monte-Carlo-Baumsuche

Inhalt, Motivation

- Ideen hinter aktuellen Go-Programmen (AlphaGoZero)

  - upper confidence bound (UCB) (ca. 1985)
  - Monte-Carlo-Baumsuche (MCTS) (ab 1993)
  - reinforcement (selbstverstärkendes) learning für CNN
    zur Verstärkung der MCTS (ab 2016)
- stärker als die besten professionellen Go-Spieler (4:1 gegen Lee Sedol 2016, 3:0 gegen Ke Jie 2017)
- das wurde lange für unmöglich gehalten
- *ohne* (Zero) Go-Expertenwissen beim Programmieren
- das ist einer der wenigen meßbaren Erfolge der KI
- bisher nicht reproduzierbar wg. hohen Rechenaufwands

Spielstärken-Bezeichnungen im Go

- historisch: Schüler (Kyu)- und Meister(Dan)-Grade:
  (schwach) 25 k $< \cdots <$ 1k $<$ 1d $< \cdots <$ 9d (stark)
  in der aktuellen Bundesliga: zehn 6d, ein 7d `https://www.dgob.de/wettbewerbe/bundesliga/mannschaften/`
- für die Spielstärken gilt idealerweise:
  bei Abstand x (Bsp: 2d gegen 2k: $x = 3$) gibt der stärkere x Vorgabe-Steine (handicap) und gewinnt mit 50 %.
  (0 Vorgaben: Weiß (zweiter) bekommt 5 Punkte (Komi))
- Grade für professionelle Spieler: 1p $< \cdots <$ 9p
  7d $\approx$ 1p, 1 Δp $\approx$ 1/3 Δd, Einstufung durch Berufsverbände (Japan, China, Korea, Europ. Go Fed.)
- Gnugo (1999–): $\approx$ 10k, Katago (2023): 9d auf OGS

Klassische Spielbaum„suche“

- es ist keine *Suche* (nach einem bestimmten Knoten), sondern eine *Bewertung* (des gesamten Baumes)
- vollständige Bewertung: (Next/Previous player wins)
  $N(x) := \exists y : (x \to y \wedge P(y))$; $P(x) := \neg N(x) = \forall y : \ldots$

- $v : G \to \{-1, +1\}$ mit $v(x) = 1 \iff N(x)$
  $v(x) = \max\{-v(y) \mid x \to y\}$, $\max \emptyset = -1$
- approximierte Bewertung v' mit Heuristik $h : G \to [-1 \dots 1]$ für entfernte und ruhige Knoten
- α/β-Verfahren: Teilbäume vermeiden, die v' nicht ändern
- für Schach erfolgreich (h bewertet Material (Figuren) u. Einfluß (bedrohte Felder), geringer Ausgandsgrad von $\to$)
  Alan Turing 1948, Deep Blue (vs. Kasparov 1996)
- für Go: keine einfache Heuristik, hoher Grad

UCB: Exploration und Exploitation

- gegeben: Spielautomaten $G_1, \dots, G_n$ mit unbekannten Erwartungswerten (für Auszahlung minus Einzahlung) (mglw. einige davon > 0) und Varianzen

  gesucht: Strategie für maximalen Gewinn

- Lösung: betätige immer den Automaten, für den oberes Ende des Vertrauensbereiches (upper confidence bound)

$$U_i = \frac{\text{Gewinn}_i \text{ bisher}}{\text{Anzahl}_i \text{ bisher}} + \frac{1}{\sqrt{\text{Anzahl}_i \text{ bisher}}}$$

  maximal ist

- T. Lai, Herbert Robbins: *Asymptotically efficient adaptive allocation rules*, 1985, `https://doi.org/10.1016/0196-8858(85)90002-8`

Monte-Carlo-Baumsuche (MCTS) für Go

- Idee: ersetze *vollständige* (min-max, alpha-beta) Baumsuche durch *stochastische* Baumsuche

  Bernd Brügmann: *Monte Carlo Go*, 1993, `http://www.ideanest.com/vegos/MonteCarloGo.pdf`

- möglichst viele lineare (nicht verzweigende) *playouts*:

  in der Wurzel beginnend, in jedem Knoten Nachfolger *nach UCB-Kriterium* auswählen

  Remi Coulom: *Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search*, CG 2006, `https://www.remi-coulom.fr/CG2006/`

- Vorteile: leicht zu implememententieren, flexibel bzgl. Zeit

  Nachteil: Initialisierung neuer Knoten ist unklar/teuer

MCTS Beispiel-Implementierung

- `https://gitlab.imn.htwk-leipzig.de/waldmann/ki-ss23/tree/master/alpha-0`

  wie besprochen, aber nur für Gomoku (statt Go)

- playout:
  - innerer Knoten: select (nach UCB)
  - äußerer K.: expand and evaluate (heuristisch/exakt)
  - backup
- Zugvorschlag in der Wurzel: nach Häufigkeit
- DAG statt Baum: jeden Knoten in Transpositionstafel (Hashtabelle) nach Zobrist 1969 `https://www.cs.wisc.edu/techreports/1970/TR88.pdf`

  Hashwert bleibt bei Zugumstellungen (Transp.) gleich!

Alpha(Go)Zero: MCTS und CNN

- David Silver et al., *Mastering the Game of Go without Human Knowledge*, Nature 2017, `https://deepmind.com/blog/alphago-zero-learning-scratch/`
- MCTS *mit CNN zur Initialisierung neuer Knoten*:
  - Eingänge: Belegung des Spielbretts ($+$ 4 Züge zurück)
  - Ausgänge: $(v, \vec{p})$
    - v: Schätzung des Spielwertes
    - $\vec{p}$: für jeden möglichen Zug eine Wahrscheinlichkeit
- aktuelles Netz wird trainiert mit (einigen) während der Playouts (mit bisher bestem Netz) bestimmten Werten
- Turnierspiele zw. aktuellem und bisher bestem Netz, Austausch bei Gewinnrate $> 55\%$

Einzelheiten zu AlphaGoZero

- extern implementiertes Spielwissen (nur) für: Erzeugen und Ausführen gültiger Züge (z.B. Fangen von Steinen), Bewerten von Endknoten (Resultat, Punktevergleich).
- $\Rightarrow$ leicht übertragbar auf andere Spiele (Schach, Shogi)
- NN minimiert die Fkt $(z - v)^2 - \pi^T \log \vec{p} + c|\theta|^2$

  wobei (z, π): Spielwert und Häufigkeiten in MCTS
- NN mit insg. 40 oder 80 Schichten (Konvolutionen, lineare Normalisierung, nichtlineare Norm. auf $[0, 1]$)
- lineare Normalisierung: Ioffe und Szegedy `https://arxiv.org/abs/1502.03167`, 2015
- ausgeführt auf Google TPUs (die Werbung dafür ist die zugrundeliegende ökonomische Motivation)

Aktuelle Entwicklungen Computer-Go (2018)

- open-Source-Nachbauten von AlphaGo und Versuch, CNN auf verteilten Privat-Rechnern zu lernen, z.B.

  Gian-Carlo Pascutto: `https://github.com/gcp/leela-zero`

  Interview (2018) `https://www.eurogofed.org/?id=205`
- Veröffentlichung von Go-Programmen großer Konzerne
  - (Mai 2018) `https://github.com/Tencent/PhoenixGo`
  - (Mai 2018) `https://research.fb.com/facebook-open-sources-elf-opengo/`
- Tencent World AI Weiqi Competition, Juni 2018 `http://computer-go.org/pipermail/computer-go/2018-June/010897.html` (FineArt 7, LeelaZero 6, ELF 4)

Aktuelle Entwicklungen Computer-Go (2023)

- open-Source-Programme, z.B. Katago

  David J. Wu *Accelerating Self-Play Learning in Go*, 2020 `https://arxiv.org/abs/1902.10565`, `https://github.com/lightvector/KataGo`

- Tony Wang, Adam Gleave et al. (2023)

  Adversarial Policies Beat Superhuman Go AIs

  `https://arxiv.org/abs/2211.00241,https://goattack.far.ai/game-analysis#contents`

Aufgaben

1. Welches Komi, welche Zeit-Regelung werden in der Bundesliga verwendet? oder auf anderen aktuellen Turnieren?

   Was ist Byoyomi?

2. bestimmen Sie die Rang-Differenz zwischen Katago, Gnugo und Ihnen selbst (wieviele Vorgaben führen zu 50% Gewinn) auf Brettgrößen 19, 13, 9.

3. Zur genaueren Messung der Spielstärke dient das ELO-System. Wie ist die Spielstärke nach ELO spezifiziert? Wie ist die Änderung der ELO-Zahl (durch Resultate von Turnier-Spielen) implementiert? Welche (empirische) Korrelation besteht zu kyu/dan-Graden?

4. die Netz-Struktur aus der Katago-Beschreibung im Quelltext wiedererkennen

5. zu Quelltext `ki-ss23/alpha0`:

   - explizite Heuristik verbessern (Schätzung für Spielwert hinzufügen, ist derzeit immer 0)
   - Heuristik aus Emacs-Gomoku übernehmen (gewichtete Anzahl der einfarbigen Blöcke)
   - (Projekt) Heuristik durch NN realisieren, lernen.

9 Analyse von Audio-Signalen

Grundlegende Methode, Anwendungen

- Spektrogramm s : Zeit $\times$ Frequenz
- Feststellung und Vergleich von elementaren musikalischen Merkmalen (Tempo, Takt, Tonhöhen, –Längen, Akkorde)

- Zerlegung in musikalisch sinnvolle Bestandteile, für:
  - Veränderung der Bestandteile, Neukombination
  - ohne Veränderung: ist es Kompression
- Quelle: M. Müller: *Fundamentals of Music Processing*, Springer 2015, 2021 (aus Kap. 2, 7, 8)

Signale, Spektrogramme

- Def: Zeit $= \mathbb{T}(= \mathbb{R})$, (Audio-)Signal: $f : \mathbb{T} \to \mathbb{R}$
- Frequenz $\mathbb{F}$, Spektrum (zu einem Zeitpunkt): $\mathbb{F} \to \mathbb{R}$,
- Ton: Summe aus Grundschwingung (Frequenz f)
  und Oberwellen (Frequenzen $2f, 3f, \dots$)
- Intensitäten der Oberwellen ergeben sich physikalischen Eingeschaften des (akustischen, elektrischen) Instrumentes und bestimmen die Klangfarbe,
- Spektrogramm: $s : \mathbb{T} \times \mathbb{F} \to \mathbb{R}$
  ist nützlich für Analyse, denn Summen von harmonischen Schwingungen haben unterschiedlichste Wellenformen, jedoch einfache Spektrogramme

Spektral-Analyse (DFT)

- grundsätzlich (Wdhlg): Darstellung einer periodischen Funktion $f : [0, 1] \to \mathbb{C}$ als $\sum_{k\in\mathbb{N}} c_k(t \mapsto \exp(2\pi ikt))$
- Anpassung für zeitdiskrete Signale (d.h., Vektoren) $x : [0, 1 \dots n-1] \to \mathbb{C}$
  Darstellung (Dekodierung) $x_t = (1/\sqrt{n}) \sum_{k=0}^{n-1} c_k \exp(2\pi ikt/n)$
  $x = M \cdot c$ mit Matrix $M_{t,k} = (1/\sqrt{n}) \exp(2\pi ikt/n)$
- Koeffizientenvektor bestimmen aus $M^{-1}x = c$
  Satz: $M^{-1}_{t,k} = (1/\sqrt{n}) \exp(-2\pi ikt/n)$
  Bew: $(M^{-1} \cdot M)_{p,r} = (1/n) \sum_q \exp(-2\pi ipq/n) \exp(2\pi iqr/n)$
- Kodierung: $c_k = (1/n) \sum_{l=0}^{n-1} x_l \exp(-2\pi ikl/n)$

DFT, FFT, Spektrogramm

- Implementierung: $M \cdot c$ (Matrix mal Vektor) schneller ausrechnen unter Ausnutzung der Struktur von M

  (fast Fourier transf., Cooley und Tukey 1965, Gauß 1805)

  `https://gitlab.imn.htwk-leipzig.de/waldmann/cm-ws22/tree/master/dft`

- Signal in Blöcke zerlegen

  (z.B. Samplerate 44.1 kHz, jeder Block 512 Samples, Blocklänge ist dann 12 ms, Blockfrequenz 86 Hz)

- DFT für jeden Block einzeln, ergibt Funktion

  Block-Index $\times$ Frequenz-Index $\to$ Koeffizient (in $\mathbb{C}$)

  (Spektrogramme in sonic-visualiser)

Spektralzerlegung zur Kompression

- die exakte DFT liefert nach Dekodierung das Original-Signal (denn $M^{-1} \cdot M = \mathrm{Id}$)
- wenn man im kodierten Signal Information entfernt (Darstellung mit Gleitkommazahlen mit wenig Bits), wird das daraus dekodierte Signal verfälscht,
  aber physiologisch (menschlich wahrnehmbar) weniger schlimm als bei entsprechendem Informationsverlust für das Original-Signal (Wellenform)
- verlustbehaftete Kompressionsverfahren MP3, AC3:
  dynamische Verteilung der vorhandenen Bitrate auf die physiologisch wichtigen Frequenzen
- typische Auswirkung: verschmierte Höhen (Hi Hats) bei fast jedem Video oder Audio-Stream

Audio Fingerprinting

- Avery Wang: *An Industrial Strength Audio Seach Algorithm*, ISMIR 2003, `http://www.ee.columbia.edu/~dpwe/papers/Wang03-shazam.pdf`

- Musikstück $\to$ Spektrogramm $\to$ Fingerabdruck (FP)

- Vergleich des FP eines unbekannten Musikstücks mit (vielen) bekannten FP aus Datenbank

- gewünschte FP-Eigenschaften: temporally localized, translation invariant, robust, sufficiently entropic
- Implementierung: Spektrogramm-Spitzen, Ankerpunkte, FP ist Menge der Differenz-Vektoren zu nahen Punkten.
- schnelle Erkennung von Diagonalen im Diagramm (Scatterplot) der Diff.-V. in Anfrage/in Datenbankeintrag

Musikalische Merkmale im Spektrogramm

- Tempo: aus regelmäßigen Intensitätsmaxima tiefster Frequenzen (Fußtrommel),
- Rhythmus: damit korrelierte regelmäßig wiederkehrende andere Geräusche (Snare, Becken)
- (Grund)Ton: Intensitätsmaximum einer Frequenz f, die keine Oberwelle ist (d.h., gleichzeitige $f/2, f/3, \ldots$ fehlen/sind schwach)
- Chromagramm (für die chromatische 12-Tonleiter $L = [C, C\sharp = D\flat, D, D\sharp = E\flat, \ldots, A\sharp = B\flat, B]$)
  für jedes $n \in L$ die Summe der Intensitäten aller Oktaven von n (Frequenzen $2^k \cdot f_n$, mit $f_n = C \cdot 2^{n/12}$)
- Akkord: 3 (oder 4) stärkste Klassen im Chromagramm
- open-source-Implementierung: sonic visualizer

NMF-basierte Signalzerlegung

- NMF = nichtnegative Matrix-Faktorisierung

  siehe M. Müller *Fundamentals of Music Proc.* 8.3 ff
- beschreibt die Erkennung und Ausnutzung von wiederholten Spalten(-Bestandteilen) im Spektrogramm

  (Spalte = Klang zu einem Zeitpunkt)
- Modell: schmales Spektrogramm $K \in F \times T'$,

  für jeden Zeitpunkt wird nichtnegative gewichtete Summe von Spalten von K gewählt durch Matrix $G \in T' \times T$,

  für das Original-Spektrogramm $S \in F \times T$ gilt $K \cdot G \approx S$.
- das ist Kompression (Abstraktion), wenn $|K| + |G| \ll |S|$

  gute Kompression erkennt Wiederholungen, also Struktur

- Einträge von G und K durch (stochastische) Optimierung

Vergleich NMF und NN

- Ziel: für gegebenes S bestimme K, G mit $K \cdot G = S' \approx S$.
- man kann K und G als voll verbundene Schichten eines NN auffassen: mit Eingabe 0 (ein Skalar); Parameter: K, G; Ausgabe S'; und nur einem Trainingsdatum S.
- Variante: in jedem Schritt ist nur eine (zufällig gewählte) Spalte von G variabel (und alles aus K)

  viele andere Varianten sind denkbar, darunter auch:
- Variante: 1. K fest, G variabel (verbessern) 2. G fest, K variabel, 3. K fest, G variabel, ...
- Beding. $K_{i,j} \geq 0, G_{i,j} \geq 0$ erschweren Gradientenabstieg.

Die multiplikative Schrittregel für NMF

- für Zielfunktion $Q = \sum_{i,j}(K \cdot G - S)^2_{i,j}$

  Gradient nach Parametern $K_{i,j}$ ausrechnen (für Parameter $G_{i,j}$ entsprechend)

  $\partial/\partial_{K_{ij}} \sum (KG - S)^2 = \ldots$
- nach Lee und Seung *Algorithms for Non-negative Matrix Factorization*, NIPS 2000: Schrittweite so wählen, daß Schritte (von K zu K', von G zu G') :

  $$K'_{i,j} = K_{i,j} \cdot \frac{(S \cdot G^T)_{i,j}}{(K \cdot G \cdot G^T)_{i,j}}, \quad G'_{i,j} = G_{i,j} \cdot \frac{(K^T \cdot S)_{i,j}}{(K^T \cdot K \cdot G)_{i,j}}$$
- Ü: was passiert hier, wenn $KG = S$, also Ziel erreicht?

  Ü: ein Bsp. für $S \in 2 \times 2, K \in 2 \times 1, G \in 1 \times 2$ ausrechnen

Beispiele Audiosignalzerlegung mit NMF

- Implementierung `https://gitlab.imn.htwk-leipzig.de/waldmann/computer-mu/-/tree/master/struct`

```
# installieren, anwenden:
cabal install --allow-newer -w /opt/ghc/ghc-9.2.7/bin/ghc
audio-struct Filter 4 0.1 exp/combined-1/input.wav
# einzelne Spuren betrachten
sonic-visualiser exp/combined-1/input/4/input-0.wav
# Summen-Signal herstellen
sox -m  exp/combined-1/input/4/input-*.wav combined-1-4.wav gain 4
```

- Beispiele: Erkennung von:
  - Wiederholung eines Akkords (Rang 1),
  - Separation von zwei Akkorden (Rang 2)
  - Isolation von 4 einzelnen Trommeln (Rang 4)

Aufgaben

1. probieren Sie (extrem) geringe Bitraten für Audio-Repräsentation mit `ffmpeg`
   (siehe manpage, Option `-b:a 128k` o.ä.)
   mit einer verlustfrei kodierten Datei beginnen (wav, flac)!
2. NMF: Gradienten für $S \in 2 \times 2, K \in 2 \times 1, G \in 1 \times 2$ mit Computeralgebra-System nachrechnen

   ```
   $ rlwrap maxima
   s:matrix([s11,s12],[s21,s22]);
   k:matrix([k11],[k21]);
   g:matrix([g11,g12]);
   d:s-k.g;
   t: (d[1,1]^2+d[1,2]^2+d[2,1]^2+d[2,2]^2);
   expand(diff(t,k11));
   ```

3. NMF: für $S = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ (oder andere Zahlenwerte)
   mit Verfahren von Lee und Seung:
   Matrizen $K \in 2 \times 1, G \in 1 \times 2$ bestimmen mit $KG \approx S$.
   Beginnen mit $K^{(0)} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, G^{(0)} = \begin{pmatrix} 1 & 1 \end{pmatrix}$

4. für NMF mit Implementierung `computer-mu/struct`: den Testfall `test2` ausführen und erklären

   ```
   $ cabal repl exe:audio-struct --allow-newer  -w /opt/ghc/ghc-9.2.7/bin/ghc
   ghci> test2
   ```

   den Parameter `rank` ändern. Für welchen Rang ist Fehler 0 (also $S = K \cdot G$) für diese Testdaten erreichbar?

10 Text-Modellierung und -Generierung

Ergänzen/Vorhersagen von Folgen (Spezif.)

- Eingabe
  - (Trainingsdaten): Text(e) $D_i \in \Sigma^*$,
  - (Prompt): $p \in \Sigma^*, p \in \Sigma^* \square \Sigma^*$

  Ausgabe
  - (nächstes Element, fehlendes Element): $c \in \Sigma$,

  so daß bedingte Wsk (c nach/in p) wie in D.
- Bsp: $D = abcacbbcaacb$, $P_D(ca|a) = 1/2, P_D(ca|b) = 0, P_D(ca|c) = 1/2$
  $P_D(a\square c|a) = 1/2, P_D(a\square c|b) = 1/2, P_D(a\square c|c) = 0$.

Ergänzen/Vorhersagen von Folgen (Impl.)

- naive Implementierungen:

  D komplett abspeichern und in jedem Schritt alle Vorkommen von p suchen.

  für alle bedingten Wsk $P(D, p|c)$ abspeichern für $p \in \Sigma^k$) (geeignete Datenstruktur: Trie)
- realistische Implementierungen

  weniger Parameter, d.h., Kompression (Informationsverlust, Abstraktion)

  mit der Hoffnung auf Vorhersagen für $p \notin D$ (Halluzinationen)

Folgen-Vorhersage durch Markov-Kette

- stochastischer Automat mit Alphabet Σ, Zustandsmenge Σ^k, Übergang $q = c'w \xrightarrow{c} wc$
- ```
  type M = M.HashMap T.Text (Transition Char)
  data Transition c = Transition
    { total :: Natural
    , next :: (M.HashMap c Natural)
    }
  build :: Int -> T.Text -> M
  generate
    :: R.RandomGen r => r -> M -> T.Text -> T.Text
  ```

  `ki-ss23/markov`
- später: weiterer Rückgriff (k größer), aber Σ^k nicht komplett gespeichert

Vektor-Semantik

- $\Sigma = W =$ Wörter, $D =$ Folgen (Sätze) von Wörtern
- welche Wörter kommen in D häufig zusammen vor?

  diese Information ist Funktion $R : W \times W \to [0 \dots 1]$
- wir benutzen eine komprimierte Darstellung von R, die Kompression erzwingt Abstraktion, repräsentiert eine *Semantik* von W (bzgl. D)
- Tomas Mikolov et al. *Distributed Representations of Words and Phrases and their Compositionality*, NIPS 2013 `https://arxiv.org/abs/1310.4546`

  Jay Alammar: *The Illustrated Word2vec*, `http://jalammar.github.io/illustrated-word`

Einbettung von Wörtern (in Vektoren)

- $\Sigma = W =$ Wörter, $D =$ Sätze, $C =$ Kontexte $(= W)$
- Korrelationsmatrix: $R : W \times C \to \mathbb{R}$

  ($R(w, c) =$ wie häufig kommt Wort w in Kontext c vor)
- Darstellung $R = A \times B$ mit $A : W \times d \to \mathbb{R}, B : d \times C \to \mathbb{R}$.
- Matrix A heißt *embedding* (von W in $\mathbb{R}^d$), Anwendungen:
  ```

- Lücken füllen: Prompt $p = l_1 \dots l_k \square r_1 \dots r_k$, belege $\square$ durch w mit $\prod\{R(w,c) \mid c \in l_1, \dots, r_1, \dots\} \to \max$
- wenn $A(w_1) \approx A(w_2)$,
  dann haben w_1 und w_2 ähnliche Bedeutung (in D)
- wenn $A(w_1) - A(w_2) \approx A(w_3) - A(w_4)$ (Parallelogramm),
  dann verhält sich w_1 zu w_2 wie w_3 zu w_4
  $w_1 =$ Berlin, $w_2 =$ Deutschland, $w_3 = ?$, $w_4 =$ Italien

Berechnung von Einbettungen

- Ziel: $R \approx A \times B$, d.h. $R(w,c) = A(w) \cdot B^T(c)$ (Skalarprodukt)
- Verfahren: Skip-Gram mit Negative Sampling:
  positives Sample: (w,c) benachbarte Wörter aus D, negatives Sample: (w,c) nicht benachbart in D
- initialisiere A, B zufällig. dann für jedes Sample (w,c):
  bestimme $u = A(w)$ (Zeile), $v = B^T(c)$ (Spalte).
  Wenn $u \cdot v$ zu klein/groß, diese Vektoren verschieben: $(u_{i+1}, v_{i+1}) = (1-r)(u_i, v_i) + r(v_i, u_i)$
  Produkt vergrößern: $0 < r (\leq 1/2)$, verkleinern: $r < 0$
- Bsp. $u = (1/2, 1/2), v = (0,1)$, $u \cdot v = 1/2$.
  $r = 1/3$, $(1-r)(u,v) + r(v,u) = ((1/3,1/3),(0,2/3)) + ((0,1/3),(1/6,1/6)) = ((1/3,2/3),(1/6,5/6)) = (u', v')$

Aufgaben

1. Implementierung `markov`: verschiedene Eingabetexte und Kontextbreiten ausprobieren.

   Die Behandlung des initialen Prompts diskutieren. (Wenn der Prompt kürzer ist als die abgespeicherten Kontexte: welche Verteilung erhält man für die direkt folgenden Zeichen? welche wäre sinnvoll?)

2. Optimierungsverfahren für `word-vec`:

   (a) Für das Verfahren der Vektor-Verschiebung: ein Beispiel mit $r < 0$ probieren (kleine Dimension). Eine allgemeine Aussage beweisen.

(b) bei Mikolov werden neuronale Netze verwendet. Vergleichen Sie. Was ist *hierarchical softmax*?

(c) verwenden Sie NNMF nach Lee and Seung, um eine Einbettung zu bestimmen.

3. Implementierung `word-vec`:

   (a) warum steht `Dieser Satz schafft Abstand` im Trainingstext?

   (b) verschiedene Eingabetexte und Dimensionen ausprobieren. Werden ähnliche Wörter erkannt? Ähnliche Wortverhältnisse?

   (c) im Programmtext wird eine Bibliothek für k-d-Bäume benutzt: zur Lösung welcher Teilaufgabe? Wieviel Platz/Zeit benötigt/gewinnt man gegenüber einer naiven Lösung?

11 Bausteine für bessere Implementierungen

Überblick, Motivation

- Ziel (bleibt): Fortsetzung eines Prompts

  durch Wsk-Verteilung für nächstes Zeichen (Token)

- Verbesserungen (gegenüber markov, word-vec) durch

  - Transformer, mit: Attention (anstatt: Rekurrenz)
  - flexible Eingabe-Zerlegung (Subword-Tokenisation)

- Nelson Elhage et al.: *A Mathematical Framework for Transformer Circuits*, 2021. `https://transformer-circuits.pub/2021/framework/`

Quellen

- Nelson Elhage et al.: *A Mathematical Framework for Transformer Circuits*, 2021. `https://transformer-circuits.pub/2021/framework/`
- Dan Jurafsky, James H. Martin *Speech and Language Processing (3rd ed.)*, 2023 `https://web.stanford.edu/~jurafsky/slp3/`
- David Foster: *Generatives Deep-Learning*, O'Reilly (2020: Kap. 6, 9)
- Mary Phuong, Marcus Hutter: *Formal Algorithms for Transformers*, 2022, `https://arxiv.org/abs/2207.09238` ...Motivation: ...lack of scientific precision and detail in DL publications.

Eingabezerlegung (Tokenization)

- Vorhersage von Folgen—über welchem Alphabet Σ?
  - (Markov-Beispiel) $\Sigma =$ die in D vorkommenden Zeichen
  - (Vektor-Semantik) $\Sigma =$ in D vorkommenden Wörter
- Mittelweg: $\Sigma =$ Wortbestandteile (subword tokenization)

  diese nicht durch exakte linguistische Analyse (Silben, Endungen, usw.), sondern sehr einfach:
- Kodierung häufig nebeneinanderstehender Zeichen:

  das neue Zeichen c ersetzt jedes ab, die Regel $c \to ab$ ist kontextfrei, ergibt *singleton CFG*

  (Ordnung $(V, >)$, zu jeder Variablen $c \in V$ genau eine Regel $c \to x \in \Sigma$ oder $c \to ab \in V^2$, für diese gilt $c > a \wedge c > b$)

Ausgabe-Repräsentation und -Verwendung

- in jedem Schritt wird bestimmt:

  (diskrete) Wsk-Verteilung p für das nächste Token

  $p = (p_1, \ldots, p_n)$ mit $0 \leq p_i \leq 1, \sum_i p_i = 1$
- tatsächlich berechnet das Netz einen Vektor $v \in \mathbb{R}^n$,

  normalisiert durch (softmax) $S_B(v) = i \mapsto \dfrac{B^{v_i}}{\sum_i B^{v_i}}$ für $B > 1$
- dabei heißt $1/B$ die *Temperatur*
  - kalt ($1/B \to 0$): nur Maxima der v_i erhalten Gewichte $\gg 0$
  - heiß ($1/B \to 1$): alle p_i stimmen nahezu überein

$v = (1, -1, 2)$, kalt: $S_{10}(v) = (0.1, 0.001, 0.9)$, $S_2(v) = (0.3, 0.1, 0.6)$, heiß: $S_{1.1}(v) = (0.35, 0.28, 0.37)$

Transformer (Encoder/Decoder-Netze)

- ursprünglich entwickelt u.a. zur automatischen Übersetzung von Sprache A nach B

  Eingabe $(\in A) \overset{\mathsf{encode}}{\longrightarrow}$ Semantik $(\in \mathbb{R}^d) \overset{\mathsf{decode}}{\longrightarrow}$ Wsk für Token der Ausgabe $(\in B)$

- Encoder und Decoder rekurrent (`mapAccumL`) $\Rightarrow$

  Berechnung der Gradienten für Fernwirkungen: teuer (nicht parallelisierbar), oft instabil (vanishing gradient)

- Jürgen Schmidhuber: *LSTM (long short-term memory)* 1995 `https://people.idsia.ch/~juergen/`

- Alternative: anstatt linearer Rekurrenz: Zeiger (*attention heads*) zum Verweis auf (Semantik von) früheren Token,

  wenige Schichten (z.B.: 6), parallelisierbar (wg. Summe)

Attention (Baustein)

- Ashish Vaswani et al., *Attention Is All You Need*, NIPS 2017, `https://arxiv.org/abs/1706.03762`

- ein *Attention-Kopf (head)* hat Eingaben: Vektoren $x_1, \ldots, x_n \in \mathbb{R}^i$, Ausgabe: Vektor $y \in \mathbb{R}^o$

  approximiert eine lineare Transformation (Tensor), hat aber *deutlich* weniger Parameter ($\Rightarrow$ schnelleres Lernen):

  Q (query) $\in (i \times k)$, K (key) $\in (i \times k)$, V (value) $\in (i \times o)$.

- Rechung: $1 \leq j \leq n$: $s_j := \langle x_j \cdot K, x_n \cdot Q \rangle$, (evtl. normiert)

  $t := \mathsf{softmax}(s)$, $y := \sum_j t_j \cdot V \cdot x_j$

  Ausgabe ist gewichtete Summe der Wert-Vektoren, Gewichte beschreiben Passung von Query und Key.

- die Hoffnung ist, daß Q, K, V Semantik kodieren

Attention (Zusammenbau)

- mehrere Köpfe (jeder mit eigenen Parametern): Ausgabe-Vektoren werden verkettet

- danach eine einfache Feed-Forward-Schicht (Multiplikation mit Matrix) $\Rightarrow$ Dimensions-Reduktion
- nach Bedarf und Laune: residual connection (Summe mit voriger Schicht), Schicht-Normalisierung

  Fig. 10.7 aus Jurafsky et al. SLP-3
- mehrere solche Schichten übereinander (wieder jede mit eigenen Parametern)

Positions-Kodierung

- nach bisher gezeigtem Modell: alle Rechnungen sind positions-unabhängig (weil über Werte der Token summiert wird).
- zur Einbettung (der untersten Schicht?)

  wird (feste) Positions-Information hinzugefügt

  einfach: absolute Position $i = 1, 2, 3, \ldots$

  geschickter: $\sin(k \cdot \pi \cdot i), \cos(k \cdot \pi \cdot i)$
- Fig. 10.6 aus Jurafsky et al. SLP-3

Attention (Training, Erzeugung)

- Fig. 10.7 aus Jurafsky et al. SLP-3

  für jeden Trainings-Satz $x_1, \ldots, x_n$:

  berechne (das geht parallel!) Ausgaben des Netzes N

  $y_1 = N(x_1), y_2 = N(x_1, x_2), \ldots, y_n = N(x_1, \ldots, x_n)$
- Bewertung ist $\sum_i - \log y_i(x_{i+1})$ (soll $\rightarrow \min$) denn:

  y_i ist die vom Netz bestimmte Wsk-Verteilung,

  $p_i = y_i(x_{i+1})$ ist die Wsk für das richtige nächste Wort,

  Ziel $p_i \approx 1$ (richtige Vorhersage), kleinere p_i bestrafen
- Anwendung: Fortsetzung des Prompts $(x_1, \ldots, x_k)$:

  $x_{k+1} = N(x_1, \ldots, x_k), x_{k+2} = N(\ldots, x_k, x_{k+1}), \ldots$

Interpretation des Transformer-Modells

- zitiert aus: Nelson Elhage et al. *A Mathematical Framework for Transformer Circuits*, 2021 `https://transformer-circuits.pub/2021/framework/index.html`
- nach Anzahl der Attention-Schichten:
  - 0: Bigramm-Statistik (Gewichte der Einbettungs-Schicht enthalten Häufigkeiten von benachbarten Token AB)
  - 1: Bigramm (AB) und Skip-Trigramm ($A \dots BC$)
  - 2 und mehr: *attention head composition*
- Bezeichnung: *residual stream* (für Token-Pos. i): Ausgaben aller Schichten an dieser Stelle i
- Thesen: 1. attention heads are independent and additive, 2. attention heads move information (between streams)

Aufgaben Kompression (Bigramm-Kodierung)

1. Bigramm-Kodierung für Texte unterschiedlicher Sprachen (verwende z.B. `https://gutenberg.org/`)

   Bsp. Norwegisch, Finnisch, Estnisch, Isländisch, Esperanto

   welches sind häufige subword token? welche davon sind text-spezifisch, welche sprach-spezifisch?

   Vergleichen Sie häufige Bigramme/Teilwörter aus Übersetzungen des gleichen Urtextes in verschiedene Sprachen

2. Def: Größe einer (singleton) CFG $G = (\Sigma, V, S, R)$ (nur für diese Aufgabe) $|G| := \sum\{(|r| : (l \to r) \in R\}$ (Summe der Längen der rechten Seiten)

   Bsp: $G_1 = (\{0,1\}, \{S,T\}, \{S \to TT0T, T \to 01\})$, $\mathsf{Lang}(G_1) = \{0101001\}$, $|G_1| = 4 + 2 = 6$, ist besser als die triviale $G_0 = (\{0,1\}, \{S\}, \{(S \to 0101001)\})$ mit $\mathsf{Lang}(G_0) = \{0101001\}$ und $|G_0| = 7$.

   Finden Sie eine gute Kompression für $[0,1,0,0,1,0,1,0,0,1,0,0,1,0,1,0,0,1,0,1,0]$

   Zeigen Sie an einem Beispiel, daß der Greedy-Algorithmus (immer ein häufigstes Paar kodieren) nicht optimal ist.

Läßt sich der Approximationsfehler beschränken?

vgl. Bannai et al. *The smallest grammar problem revisited*, SPIRE 2016, `https://arxiv.org/abs/1908.06428`

Aufgaben Transformer

1. Anzahl der Parameter einers Transformers diskutieren (abhängig von Anzahl Schichten, Köpfen usw.)

2. Auswirkung des Temperatur-Parameters für Ausgabe, für innere Schichten beobachten.

   In der Literatur werden die Skalarprodukte (Query $\times$ Key) normiert (Division durch $\sqrt{d}$), vergleiche mit Temperatur des Softmax

3. Code ergänzen/Zweck diskutieren:

   - Positions-Kodierung (von links, von rechts, gelernt/fixiert)
   - Schicht-Normalisierung (auf Mittelwert 0, Varianz 1)
   - Beam-Suche

4. Diskutiere die Beobachtung *the location of the LayerNorm in this figure remains a hotly debated subject* aus Sebastian Raschka: *About LayerNorm Variants in the Original Transformer Paper, ...* `https://magazine.sebastianraschka.com/p/why-the-original-transformer-figure`

5. Thesen aus Elhage et al. an einfachen Beispielen überprüfen.

6. Schmidhuber et al. *Linear Transformers Are Secretly Fast Weight Programmers* `https://arxiv.org/abs/2102.11174`

7. (Forschungsaufgabe) welche (formalen) Sprachen kann man mit Transformer/Attention (Ausgabe ist eine Zahl: Wsk für $w \in L$) exakt darstellen? Darunter nicht reguläre? nicht kontextfreie? Welches ist das dazu passende (exakte) Maschinenmodell? Welche Rolle spielen dabei: die Anzahl der Köpfe? der Schichten? Kann man die darstellbaren Sprachen tatsächlich auch (asymptotisch) exakt lernen?

8. (Forschung) LSTM ist als `mapAccumL` darstellbar?

9. (Forschung) Transformer für Bäume (anstatt Folgen)

12 Einschätzung, Ausblick

Struktur (tief) oder nicht (flach)

- für die flachen Modelle (z.B. Transformer) spricht:

  daß Parameter (konzeptuell) einfach zu bestimmen sind (mit Rechenaufwand, aber der ist parallelisierbar)

- dagegen spricht: daß keine beliebig hohen (geschachtelten) Bäume realisiert werden können

- Kritik dagegen ist: na und—kann der Mensch auch nicht.

- Kritik dagegen: aber Maschinen müssen das können, z.B. Parser für Programmtexte,

  die muß dann der Mensch programmieren,

  denn eine sog. KI versteht schon die Aufgabe gar nicht

Beispiel: Presburger-Arithmetik (PA)

- Gegenstandsbereich: prädikatenlogische Formeln in Signatur: zweistelliges Funktionssymbol $(+)$, zweistelliges Relationssymbol $(=)$.

  interpretiert im Universum $\mathbb{N}$ auf die übliche Weise: $(+)$ durch Addition, $(=)$ durch Gleichheit

- Beispiele: $\exists x : \forall y : x + y = y$,

- die Relation $x \leq z$ ist definierbar ($\equiv \exists y : x + y = z$),

  ebenso $(x < z) \equiv (x \leq z) \wedge \neg(x = z)$

- gesucht ist Entscheidungsverfahren für die Wahrheit (Allgemeingültigkeit) solcher Sätze

  Bsp: $\forall x : \forall y : \neg(x < y \wedge y < x)$

PA — Entscheidungsverfahren

- historisch erste Lösung von Mojżesz Presburger (1929)

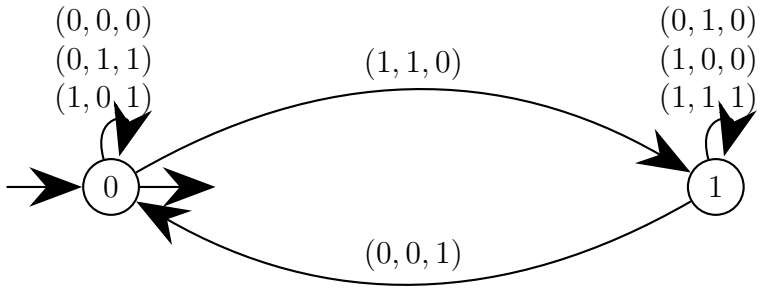
- Lösungs-Idee von J. R. Büchi (1960):

  Modellmenge $\mathsf{Mod}(F)$ für Teilformel F (mit freien Variablen $x_1, \ldots, x_k$) durch geschichtete Binärkodierung (LSB links) als Sprache $\subseteq (\mathbb{B}^k)1*$ darstellen

- Satz: jede solche Modellmenge ist regulär (durch NFA repräsentierbar)
- logische Verknüpfungen und Quantoren als Sprach-Op. implementieren,
- Formel F erfüllbar $\iff \mathsf{Mod}(F) \neq \emptyset \iff \mathsf{Lang}(A) \neq \emptyset$

PA: Addition, Gleichheit

- initial: 0, final: 0, Übergänge: $c_{\text{in}} \xrightarrow{(x,y,s)} c_{\text{out}}$ (Voll-Addierer)
- $A =$



- Bsp: $(1, 2, 3) \in \mathsf{Mod}(x + y = z)$

  $x = 1 =\ 1\ \ 0$
  repräsentiert als $y = 2 =\ 0\ \ 1$ (LSB links),
  $z = 3 =\ 1\ \ 1$

  Stellen stapeln: $(1, 0, 1)(0, 1, 1) \in \mathsf{Lang}(A)$
- Gleichheit: $\mathsf{Mod}(x = y) = \{(0, 0), (1, 1)\}^*$

PA: aussagenlogische Operationen

- realisiert durch Mengenoperationen auf Modellmengen

  $\mathsf{Mod}(F \wedge G) = \mathsf{Mod}(F) \cap \mathsf{Mod}(G)$

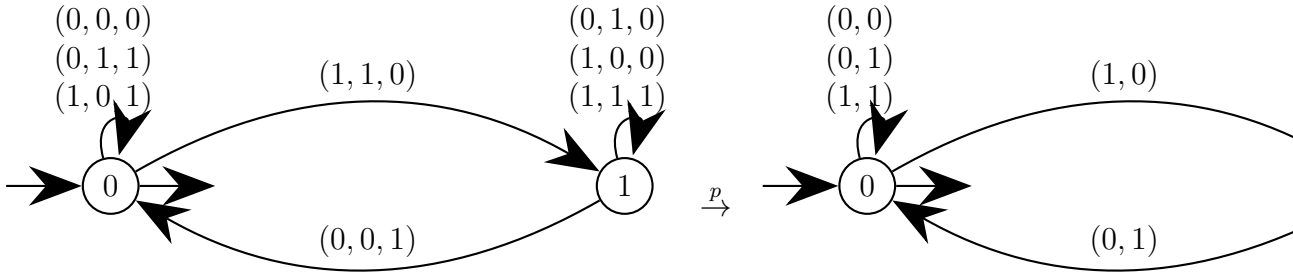
  $\mathsf{Mod}(F \vee G) = \mathsf{Mod}(F) \cup \mathsf{Mod}(G)$

  $\mathsf{Mod}(\neg F) = \Sigma^* \setminus \mathsf{Mod}(F)$ für $\Sigma = \mathbb{B}^{|\,\mathsf{fvar}(F)|}$
- Implementierung: ist einfach möglich für deterministische und vollständige Automaten

- $\cap$ und $\cup$ durch Kreuzprodukt-Konstruktion,
- Komplement durch Tauschen von finalen mit nicht finalen Zuständen

PA: Quantifikation

- $\mathsf{Mod}(\exists x.F)$ durch Projektion p von $\mathsf{Mod}(F)$
  p von Alphabet $\mathbb{B}^f$ zu Alphabet $\mathbb{B}^{f-1}$ mit $f = |\,\mathsf{fvar}(F)|$
- Bsp: links $\mathsf{Mod}(x + y = z)$, rechts $\mathsf{Mod}(\exists y : x + y = z)$



- All-Quantifikation umformen: $(\forall x.F) \equiv \neg(\exists x.\neg F)$
- Impl.: Projektion zerstört evtl. Determinismus, durch Potenzmengenkonstruktion wiederherstellen, ist teuer

PA: eine konkrete Implementierung

- Beispiel-Eingabe: Formel (in Datei `data/ex1.pb`)

```
let { le x y = exists z . x = y + z
    ; lt x y = le x y && ! (x = y)
} in forall x . forall y . ! (lt x y && lt y x)
```

- Beispiel-Rechnung:

```
cabal run presburger-arithmetic-main -- data/ex1.pb
...
Models {arity = Index 0, auto = NFA ... }
True
```

- Code: `https://gitlab.imn.htwk-leipzig.de/waldmann/presburger-arithmetic`
  benutzt NFA-Bibliothek aus autotool (liefert dafür gute Testfälle)

PA: kleine Formel, große Modelle

- für jedes $n \in \mathbb{N}$ gibt es PA-Formel G_n der Größe $\Theta(n)$ mit
  $G_n(x, y, z) \iff (x = 2^{2^n}) \wedge (x \cdot y = z)$
  (nach einer Konstruktion von Fischer und Rabin 1974)
- Induktionsanfang: $G_0(x, y, z) \equiv (x = 2 \wedge y + y = z)$
- Induktionsschritt (Ansatz) $G_{n+1}(x, y, z) \equiv \exists h : \exists u : G_n(h, h, x) \wedge G_n(h, y, u) \wedge G_n(h, u, z)$
  dann gilt aber $|G_{n+1}| \approx 3 \cdot |G_n|$, also $|G_n| \in \exp(\Theta(n))$
- der Trick ist nun: $\exists h : \exists u : \forall p : \forall q : ((p = h \wedge q = x) \vee \ldots \vee \ldots) \Rightarrow G_n(h, p, q)$
  die All-Quantoren gestatten die Nachnutzung der einen Teilformel G_n

Kann eine generative KI die Presburger-Arithmetik erfinden?

- von allein sicher nicht,
- ein Generator könnte Glück haben:
  - wenn unter den Trainingsdaten eine Beschreibung/Implementierung der PA waren,
  - und diese war korrekt

Die tatsächliche Qualität der Trainingsdaten

- entspricht der Qualität der Texte, die es eben online gibt
- dort steht überwiegend Unsinn, und dieser ist tausendfach kopiert (Bsp: Kopien von Stackoverflow, als SEO-Spam)
- echt handgeschriebene Texte gelten trotzdem als wertvoll (z.B. in Archiven von sog. sozialen Netzwerken)
- vergleiche aber Veselovsky et al., Artificial Artificial Artificial Intelligence: Crowd Workers Widely Use Large Language Models for Text Production Tasks, `https://arxiv.org/abs/2306.07899`

  R. Williams: The people paid to train AI are outsourcing their work… to AI, MIT Technology Review, 2023 `https://www.technologyreview.com/2023/06/22/1075405/the-people-paid-to-train-ai-are-outsourcing-their-work`

Auswirkungen der generativen KI

- . . . das ist schon falsch gefragt, die KI selbst bewirkt gar nichts, der Antrieb sind die wirtschaftlichen Interessen der Dienst-Anbieter.
- darauf muß die Gesellschaft kritisch aufpassen

  funktioniert in Einzelfällen auch: John Brodkin: Lawyers have real bad day in court after citing fake cases made up by ChatGPT, Ars Technica, 23. 6. 2023, `https://arstechnica.com/tech-policy/2023/06/lawyers-have-real-bad-day-in-cou`
- Aufgabe der Hochschule (der Informatik-Lehre): nicht den (behaupteten) Trends hinterherlaufen, sondern die Studenten/die Öffentlichkeit gegen diese immunisieren