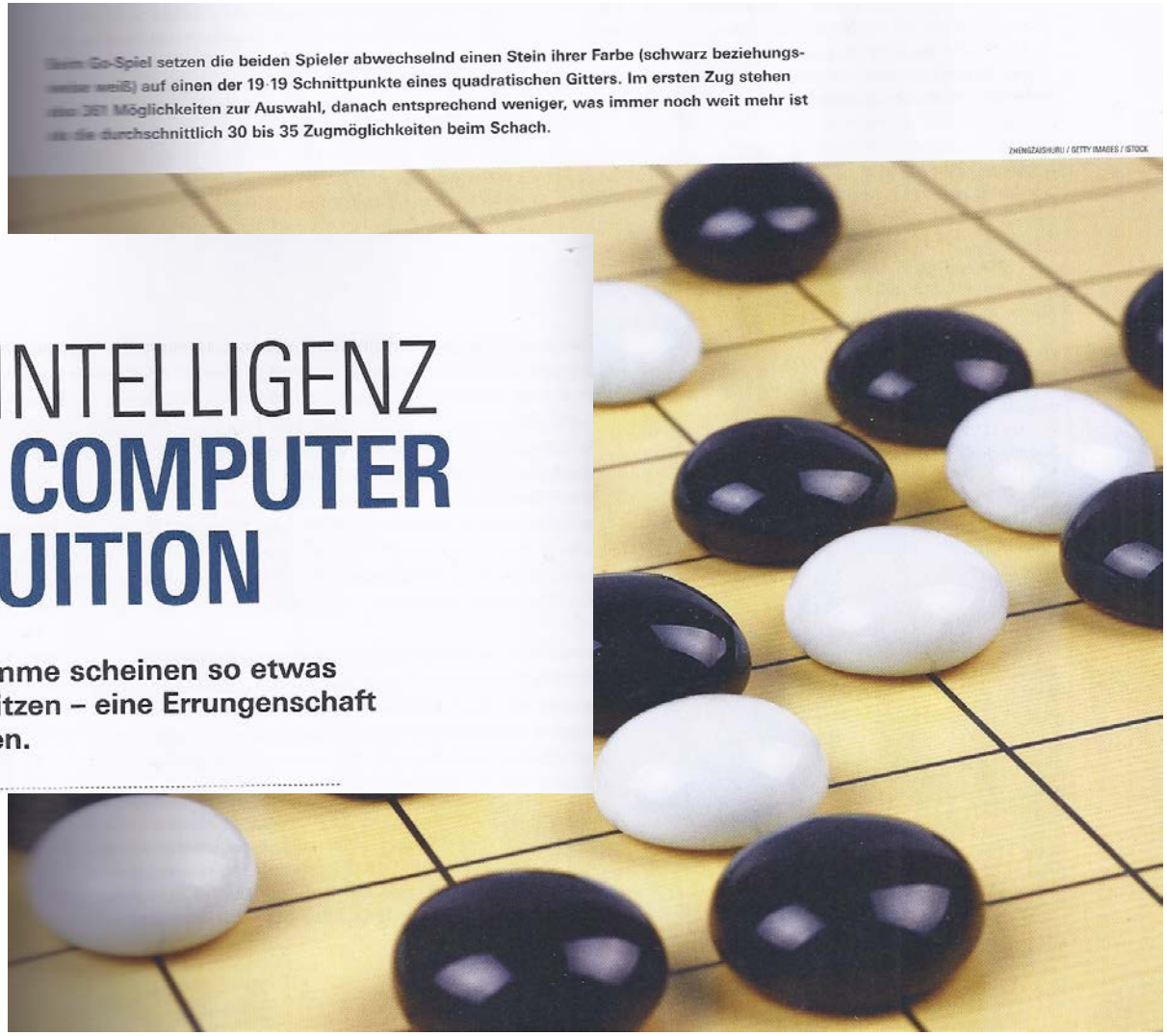
KALAHA

Erfahrungen bei der Implementation von neuronalen Netzen in APL

Dipl.Math. Ralf Herminghaus, April 2018

1. Die Schlagzeile



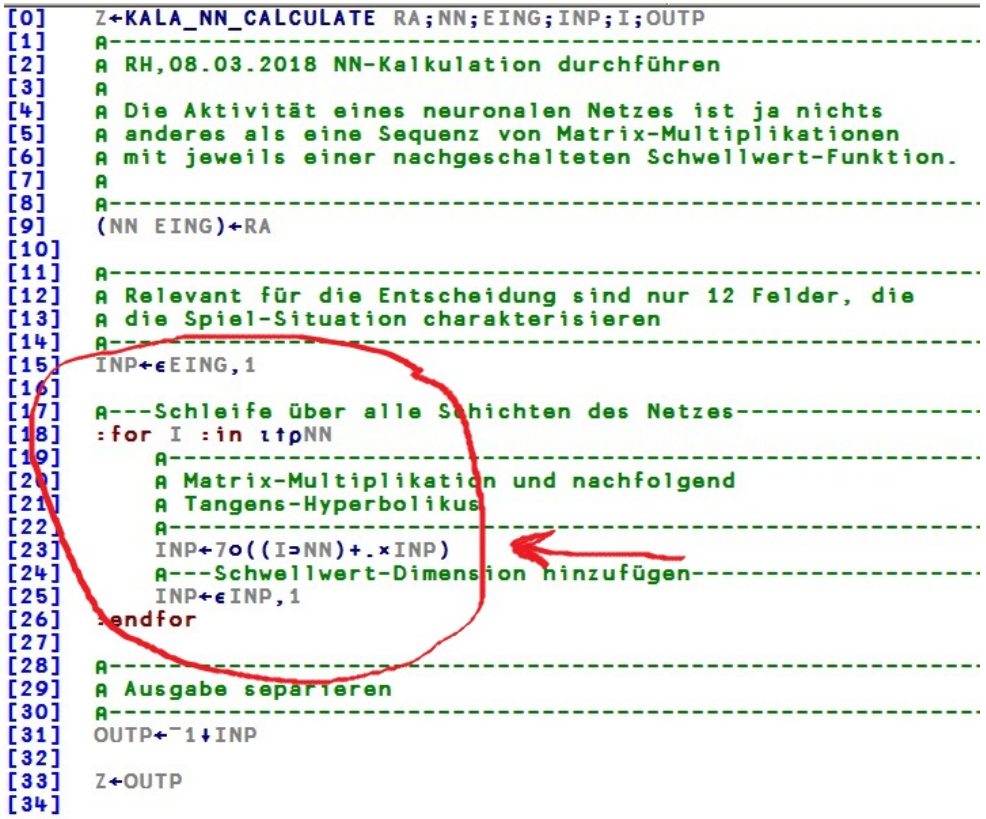
KÜNSTLICHE INTELLIGENZ

ALPHA GO – COMPUTER LERNEN INTUITION

Die neuen Go spielenden Programme scheinen so etwas wie menschliche Intuition zu besitzen – eine Errungenschaft mit weit reichenden Konsequenzen.

2. Die Idee

APL ist eine Super-Sprache!
Also: So schwierig kann das ja nicht sein ....



```
[0]   Z←KALA_NN_CALCULATE RA;NN;EING;INP;I;OUTP
[1]   ⍝-------------------------------------------------------------
[2]   ⍝ RH,08.03.2018 NN-Kalkulation durchführen
[3]   ⍝
[4]   ⍝ Die Aktivität eines neuronalen Netzes ist ja nichts
[5]   ⍝ anderes als eine Sequenz von Matrix-Multiplikationen
[6]   ⍝ mit jeweils einer nachgeschalteten Schwellwert-Funktion.
[7]   ⍝
[8]   ⍝-------------------------------------------------------------
[9]   (NN EING)←RA
[10]
[11]  ⍝-------------------------------------------------------------
[12]  ⍝ Relevant für die Entscheidung sind nur 12 Felder, die
[13]  ⍝ die Spiel-Situation charakterisieren
[14]  ⍝-------------------------------------------------------------
[15]  INP←,EING,1
[16]
[17]  ⍝---Schleife über alle Schichten des Netzes-----------------
[18]  :for I :in ⍳⍴NN
[19]      ⍝-------------------------------------------------------
[20]      ⍝ Matrix-Multiplikation und nachfolgend
[21]      ⍝ Tangens-Hyperbolikus
[22]      ⍝-------------------------------------------------------
[23]      INP←7○((I⊃NN)+.×INP)
[24]      ⍝---Schwellwert-Dimension hinzufügen-------------------
[25]      INP←,INP,1
[26]  :endfor
[27]
[28]  ⍝-------------------------------------------------------------
[29]  ⍝ Ausgabe separieren
[30]  ⍝-------------------------------------------------------------
[31]  OUTP←¯1↓INP
[32]
[33]  Z←OUTP
[34]
```

Ein neuronales Netz in APL zu implementieren, ist ja extrem einfach .....

Wo ist das Problem ?

3.Das Problem

Die Rechenleistung:

Alpha-Zero: 11,50000000000 PetaFlops

Mein Rechner: 0,00000009321 PetaFlops

➔ Vielleicht sollte ich nicht gerade mit GO anfangen .....

4.Die Alternativen

GO: ca. 350 Wahlmöglichkeiten pro Zug

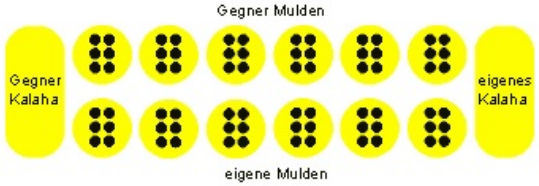
Schach: ca. 30-40 Wahlmöglichkeiten pro Zug

KALAHA: max. 6 Wahlmöglichkeiten pro Zug

➔ Ich probiere es erstmal mit KALAHA

Kalaha

Spielregeln



Startaufstellung

je 6 Kugeln werden in die 12 kleinen Mulden gelegt

Gewinner ist, wer bei Spielende die meisten Kugeln in seinem Kalaha hat.

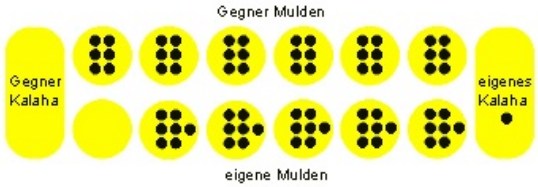
Wer am Zuge ist, leert eine seiner Mulden und verteilt die Kugeln, jeweils eine, reihum im Gegenuhrzeigersinn in die nachfolgenden Mulden. Dabei wird auch das eigene Kalaha gefüllt. Das Gegner Kalaha wird ausgelassen.

Fällt die letzte Kugel ins eigene Kalaha, ist der Spieler nochmals am Zuge.

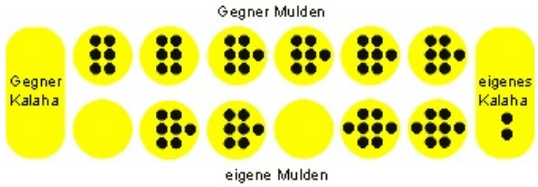
Fällt die letzte Kugel in eine leere Mulde auf der eigenen Seite, wird diese Kugel und alle Kugeln in der Gegner Mulde gegenüber, ins eigene Kalaha gelegt und der Gegner hat den nächsten Zug.

Das Spiel ist beendet, wenn alle Mulden eines Spielers leer sind. Der Gegner bekommt dann alle Kugeln aus seinen Mulden in sein Kalaha.

Beispiel:

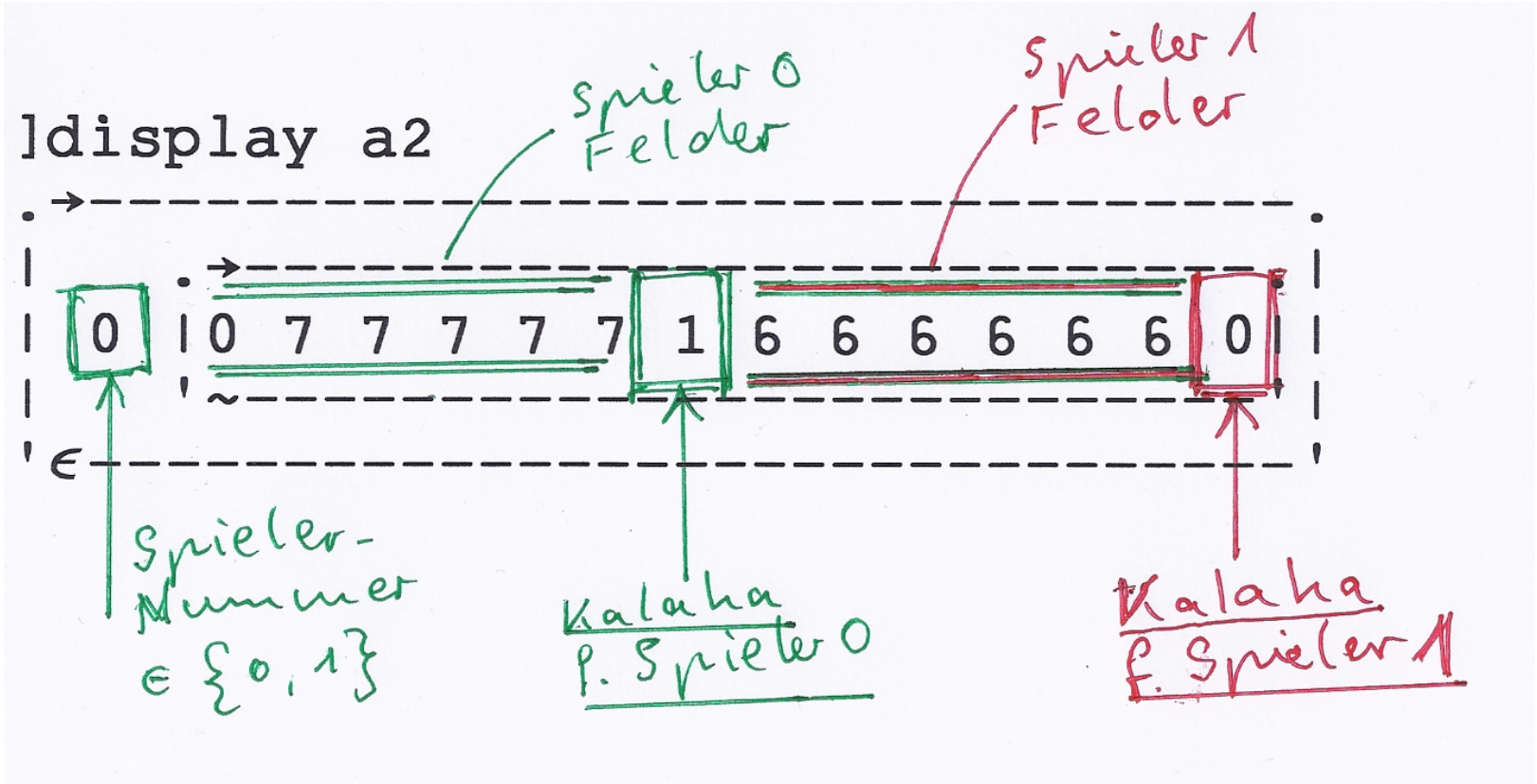


1. Spieler hat die Mulde ganz links geleert und ist noch mal am Zuge, da die letzte Kugel in seinem Kalaha gelandet ist.

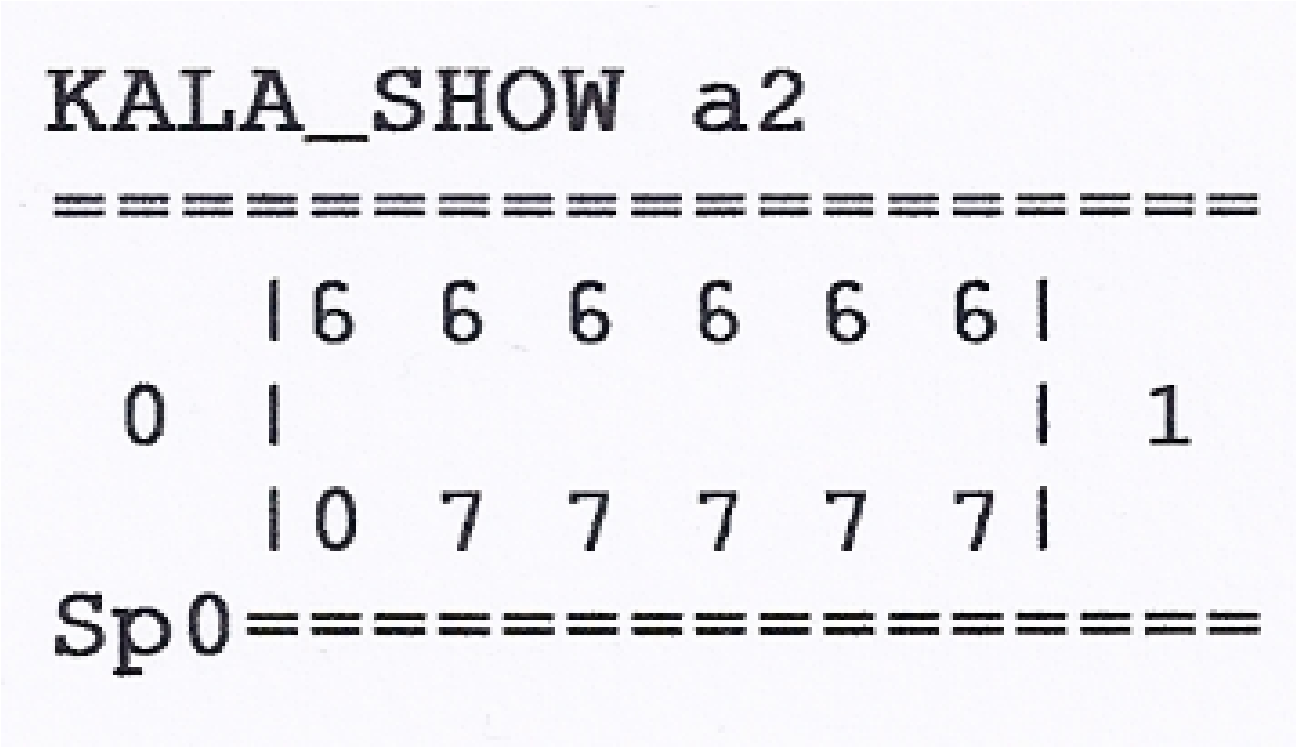


1. Spieler hat seine 4 Mulde geleert, jetzt ist der 2. Spieler am Zuge.

Wie kann man eine Spiel-Situation denn am besten in der internen Darstellung abbilden ?



Darstellung nach außen

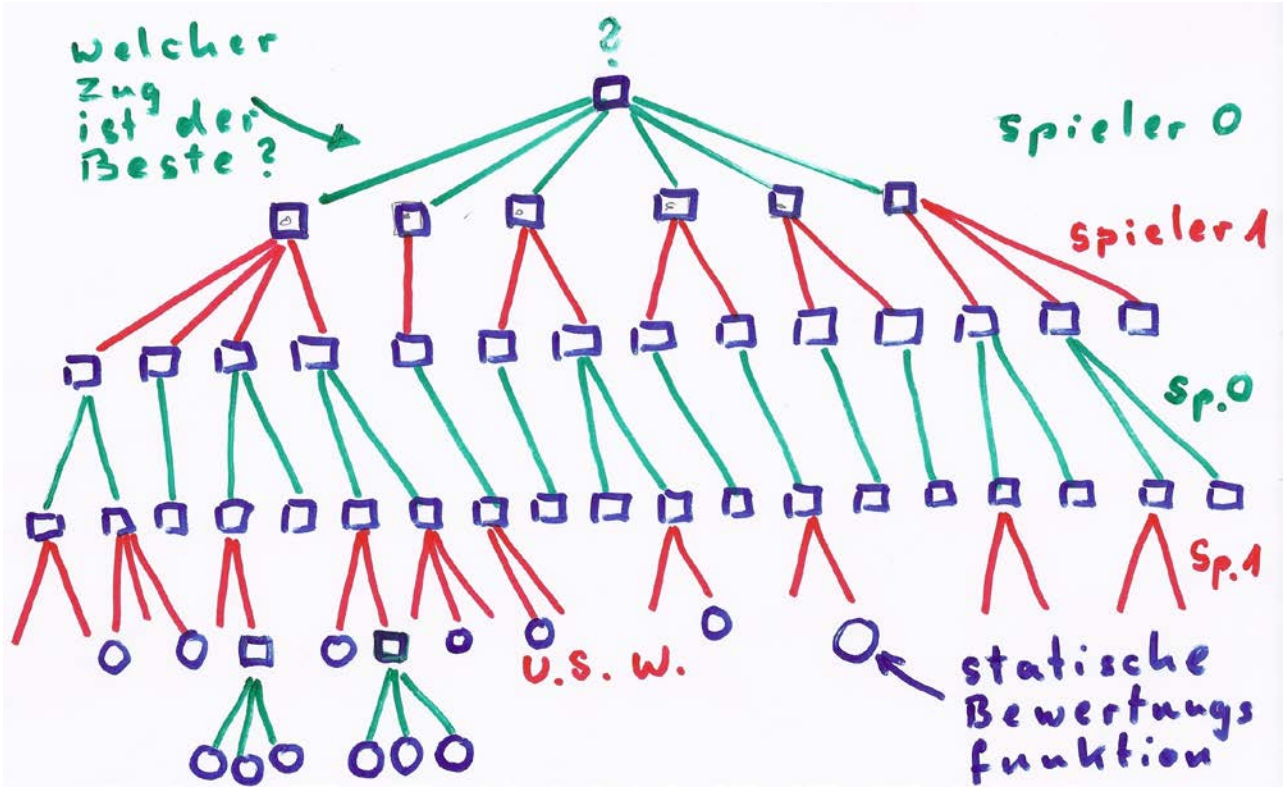


Einige Spiel-Züge als Beispiel

```
]display KALA_START
.→-----------------------------.
|   .→-------------------------.|
| 0 |6 6 6 6 6 6 0 6 6 6 6 6 6 0||
|   '~-------------------------'|
'∈-----------------------------'
a1←KALA_START
      a2←a1 KALA_ZUG 1
]display a2
.→-----------------------------.
|   .→-------------------------.|
| 0 |0 7 7 7 7 7 1 6 6 6 6 6 6 0||
|   '~-------------------------'|
'∈-----------------------------'
      a3←a2 KALA_ZUG 2
]display a3
.→-----------------------------.
|   .→-------------------------.|
| 1 |0 0 8 8 8 8 2 7 7 6 6 6 6 0||
|   '~-------------------------'|
'∈-----------------------------'
      a4←a3 KALA_ZUG 5
]display a4
.→-----------------------------.
|   .→-------------------------.|
| 0 |1 1 9 9 8 8 2 7 7 6 6 0 7 1||
|   '~-------------------------'|
'∈-----------------------------'
      a5←a4 KALA_ZUG 1
]display a5
.→-----------------------------.
|   .→-------------------------.|
| 1 |0 2 9 9 8 8 2 7 7 6 6 0 7 1||
|   '~-------------------------'|
'∈-----------------------------'
      a6←KALA_INVERT a5
]display a6
.→-----------------------------.
|   .→-------------------------.|
| 0 |7 7 6 6 0 7 1 0 2 9 9 8 8 2||
|   '~-------------------------'|
'∈-----------------------------'
```

```
KALA_SHOW a1
====================
   |6 6 6 6 6 6|
 0 |           | 0
   |6 6 6 6 6 6|
Sp0=================

KALA_SHOW a2
====================
   |6 6 6 6 6 6|
 0 |           | 1
   |0 7 7 7 7 7|
Sp0=================

KALA_SHOW a3
Sp1=================
   |6 6 6 6 7 7|
 0 |           | 2
   |0 0 8 8 8 8|
====================

KALA_SHOW a4
====================
   |7 0 6 6 7 7|
 1 |           | 2
   |1 1 9 9 8 8|
Sp0=================

KALA_SHOW a5
Sp1=================
   |7 0 6 6 7 7|
 1 |           | 2
   |0 2 9 9 8 8|
====================

KALA_SHOW a6
====================
   |8 8 9 9 2 0|
 2 |           | 1
   |7 7 6 6 0 7|
Sp0=================
```

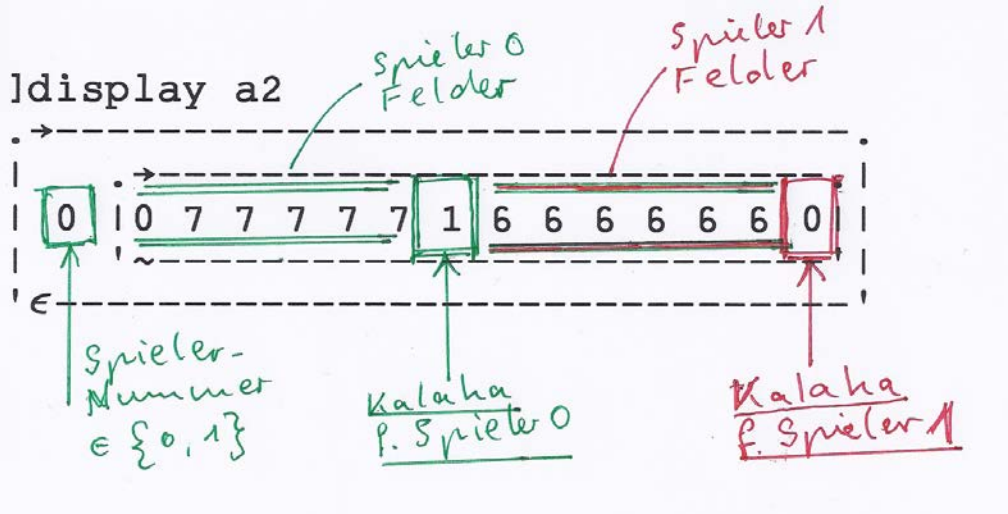
Generelles Vorgehen bei der Bewertung von Spiel-Situationen in 2-Personen-Strategiespielen



Wenn Spieler 0 einen Zug macht, welche Möglichkeiten hätte dann Spieler 1, und welche Möglichkeiten hätte Spieler 0, darauf zu antworten, usw. usw. .....

Die Statische Bewertungs-Funktion

Die „natürlichste“ Bewertungsfunktion ist die Differenz der Inhalte der beiden Kalaha-Felder:



Hier etwa:

Bew(a2) = TAB[7] – TAB[14]

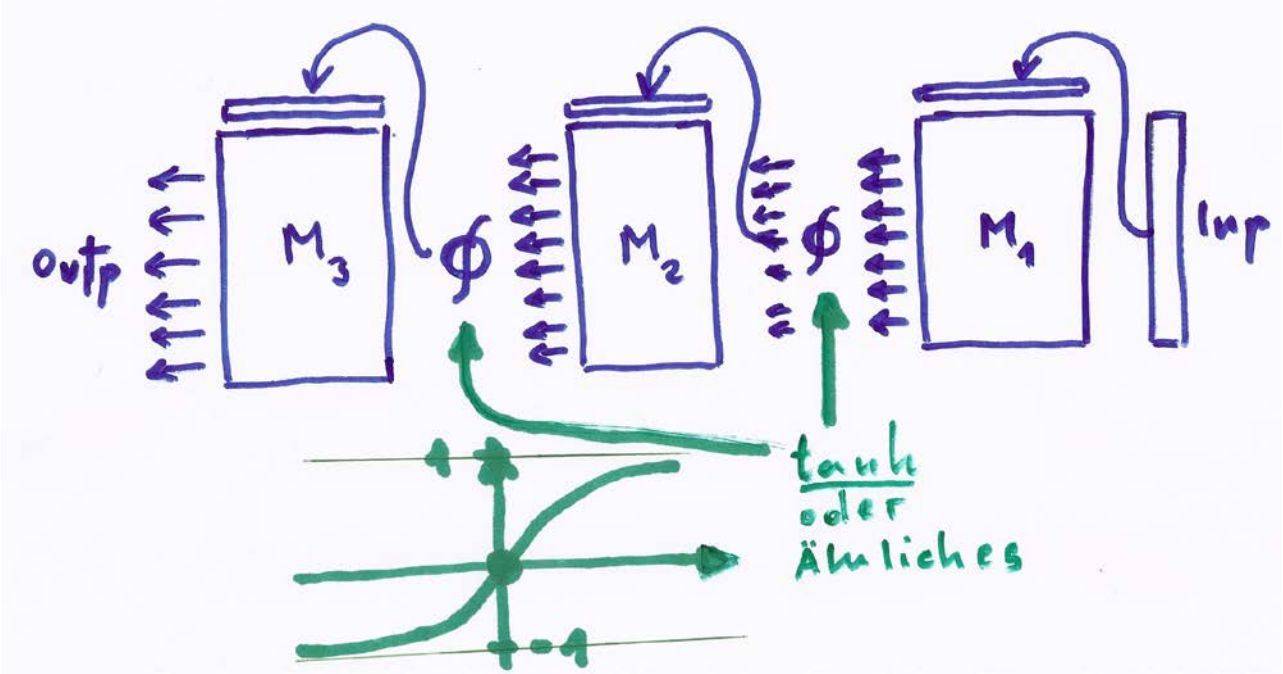
Probleme bei der rekursiven Bewertung von Spiel-Situationen

- Die Bewertungs-Funktion wird bereits bei geringen Rekursions-Tiefen extrem „teuer“, d.h. man muß auf eine entsprechende Bewertung sehr lange warten.
- Man ist also darauf angewiesen, die Such-Tiefe irgendwie einzuschränken
- Nimmt man als „statische Bewertungsfunktion“ einfach die Differenz der beiden Kalahas, so erreicht man keine wirklich befriedigende Spiel-Stärke

Mögliche Lösung des Problems

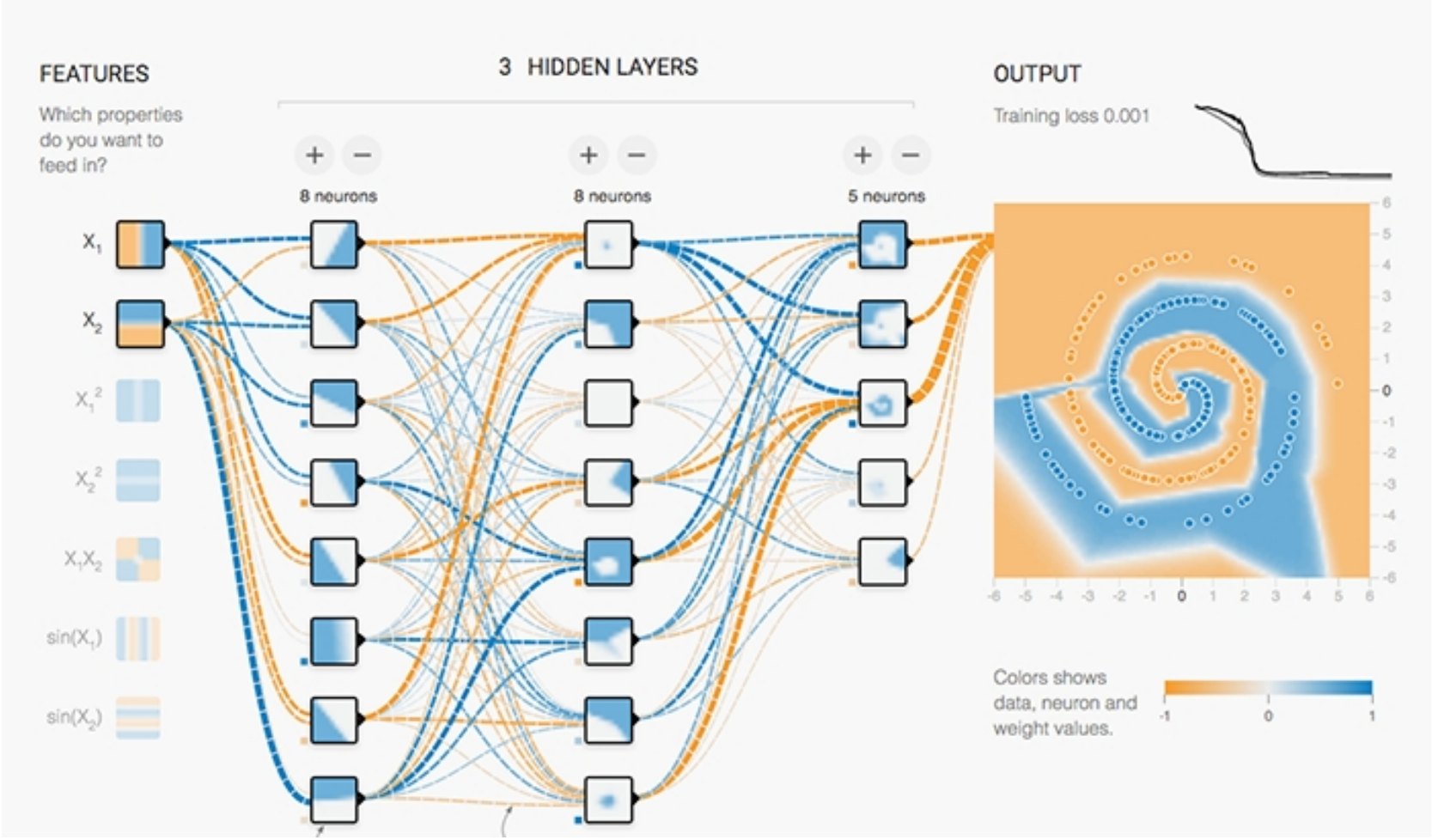
- Schränke die Suchtiefe ein, aber ....
- Ersetze die „Statische Bewertungsfunktion“ durch eine „intelligente“ Schätzung durch ein neuronales Netz

Wie funktioniert ein „Neuronales Netz“ ?



Ein einfaches „neuronales Netz“ mit n Schichten läßt sich realisieren als Sequenz von n Matrix-Multiplikationen mit eingeschobenen Schwellwert-Funktionen (Tangens - Hyperbolicus, Sigmoid-Funktion, o.Ä.) . Es gibt noch andere Typen von neuronalen Netzen – die wir hier aber nicht betrachten wollen.

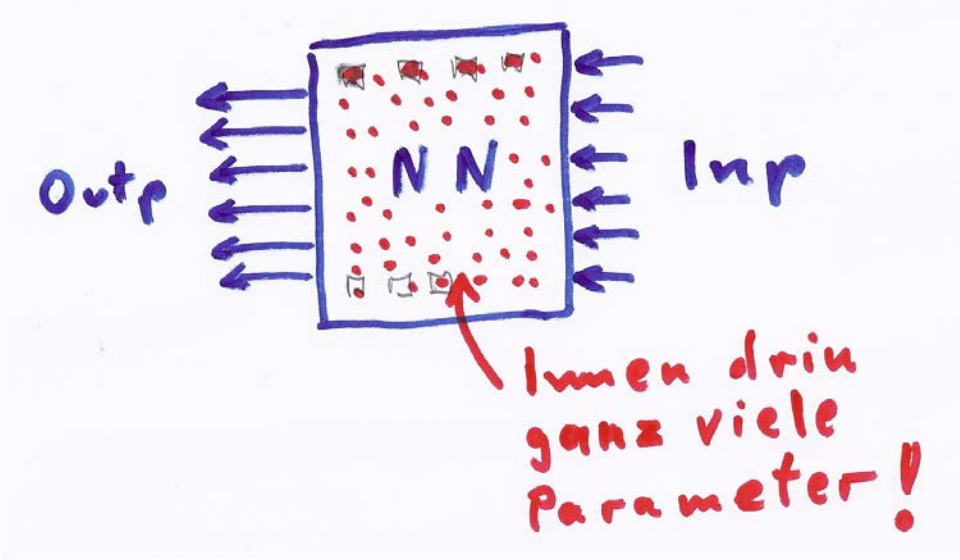
Eine andere übliche Darstellung solcher Netze



In APL sieht die Implementation eines mehrschichtigen neuronalen Netzes folgendermaßen aus:

```
[0]    Z←KALA_NN_CALCULATE RA;NN;EING;INP;I;OUTP
[1]    ⍝-------------------------------------------------------
[2]    ⍝ RH,08.03.2018 NN-Kalkulation durchführen
[3]    ⍝
[4]    ⍝ Die Aktivität eines neuronalen Netzes ist ja nichts
[5]    ⍝ anderes als eine Sequenz von Matrix-Multiplikationen
[6]    ⍝ mit jeweils einer nachgeschalteten Schwellwert-Funktion.
[7]    ⍝
[8]    ⍝-------------------------------------------------------
[9]    (NN EING)←RA
[10]
[11]   ⍝-------------------------------------------------------
[12]   ⍝ Relevant für die Entscheidung sind nur 12 Felder, die
[13]   ⍝ die Spiel-Situation charakterisieren
[14]   ⍝-------------------------------------------------------
[15]   INP←∊EING,1
[16]
[17]   ⍝---Schleife über alle Schichten des Netzes-------------
[18]   :for I :in ⍳↑⍴NN
[19]       ⍝-------------------------------------------------------
[20]       ⍝ Matrix-Multiplikation und nachfolgend
[21]       ⍝ Tangens-Hyperbolikus
[22]       ⍝-------------------------------------------------------
[23]       INP←7○((I⊃NN)+.×INP)
[24]       ⍝---Schwellwert-Dimension hinzufügen-----------------
[25]       INP←∊INP,1
[26]   :endfor
[27]
[28]   ⍝-------------------------------------------------------
[29]   ⍝ Ausgabe separieren
[30]   ⍝-------------------------------------------------------
[31]   OUTP←¯1↓INP
[32]
[33]   Z←OUTP
[34]
```

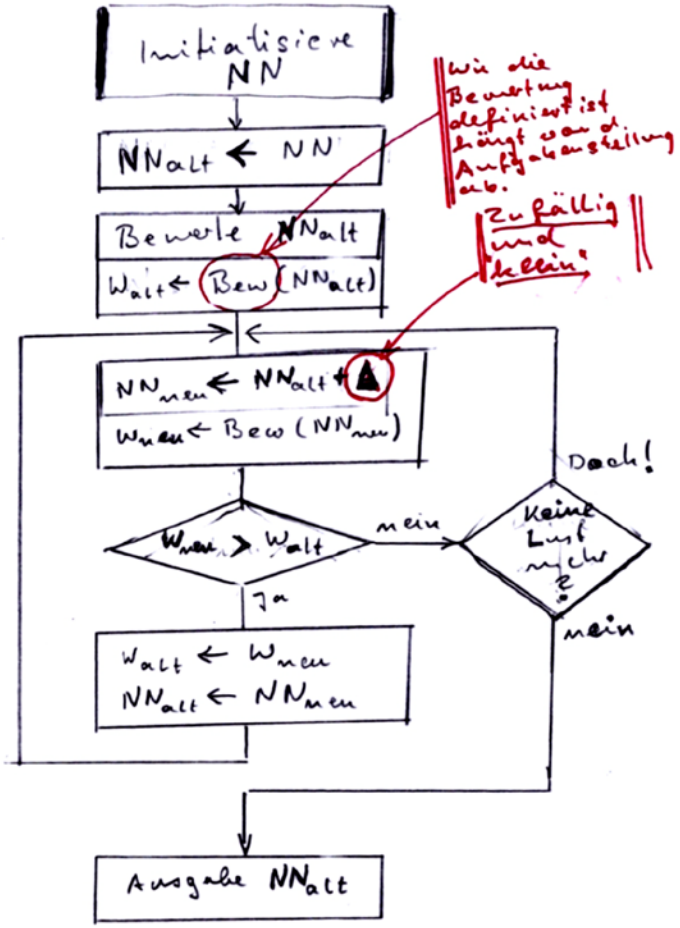
GANZ grob vereinfacht kann man es auch so sehen:



Das „Training“ eines neuronalen Netzes erfolgt einfach, indem man die Parameter (d.h. die Koeffizienten der beteiligten Matrizen) solange manipuliert, bis die gewünschte Zuordnung von Input zu Output erreicht ist.

Wenn das Netz eine große Anzahl von Beispiel-Fällen korrekt bearbeitet, kann man darauf hoffen, daß es das auch bei anderen - bisher noch nicht aufgetretenen Fällen – ebenso korrekt tut.

Einfacher Evolutions-Algorithmus zum „Training“ eines neuronalen Netzes



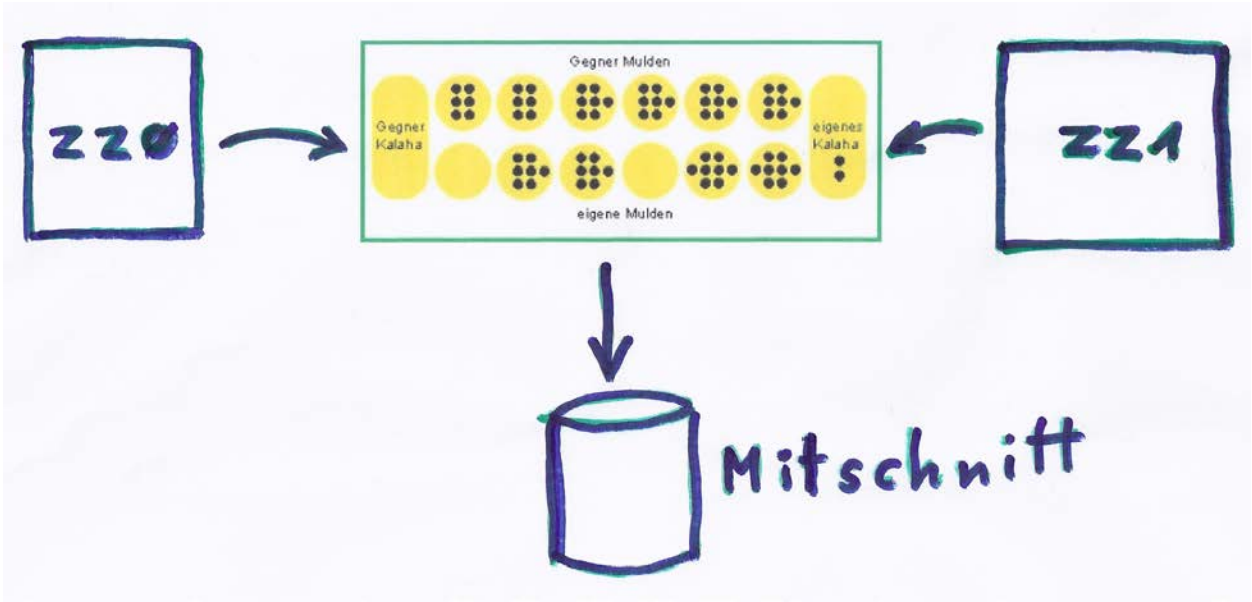
Der hier dargestellte Evolutions-Algorithmus ist extrem einfach und – leider - auch relativ ineffizient. D.h. das Training des Netzes braucht viel Zeit.

Andere Trainings-Verfahren wie z.B. „Back-Propagation“ sind deutlich effizienter, aber auch nicht in allen Fällen anwendbar.

Der Master-Plan

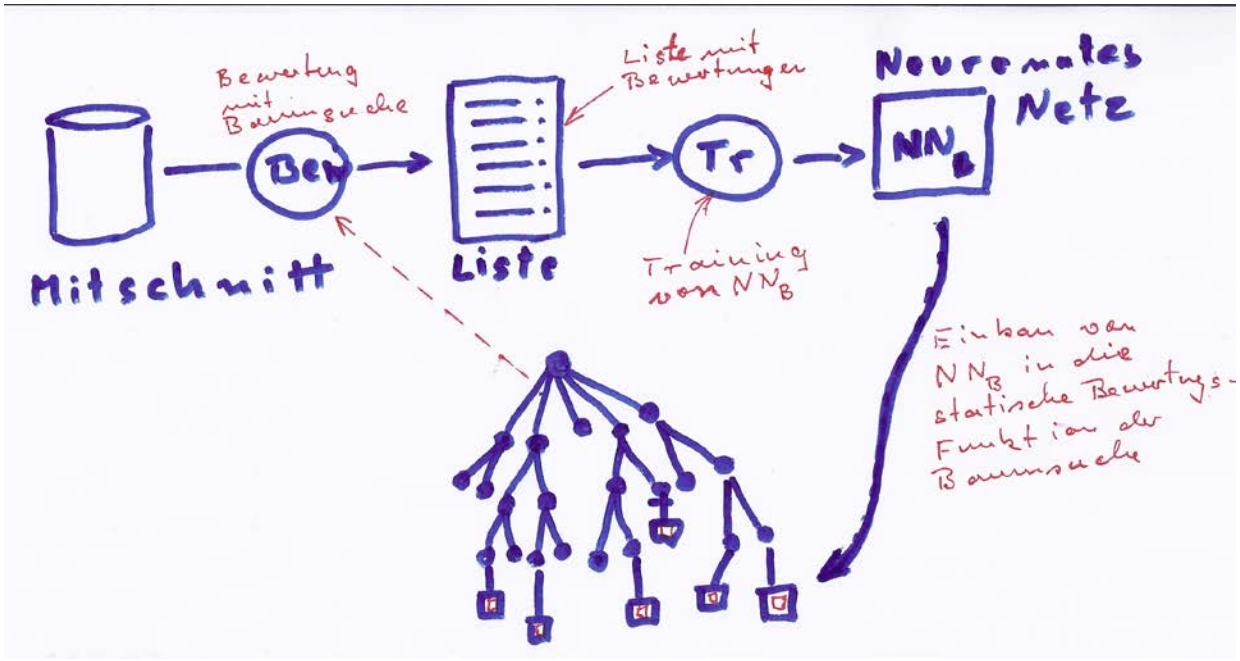
- Erzeuge zunächst eine große Menge an realistischen Spiel-Situationen (10000 Stück – oder so)
- Bewerte jede dieser Situationen mit Hilfe einer rekursiven Bewertungs-Funktion und speichere die Ergebnisse in einer Liste ab. (**Vorsicht!** Das kann lange dauern, weil die rekursive Bewertungsfunktion sehr aufwändg ist.)
- Trainiere ein neuronales Netz anhand der vorliegenden Liste darauf, die rekursive Bewertungsfunktion möglichst gut zu imitieren. (**Vorsicht!** Auch das kann lange dauern – vor allem mit dem oben beschriebenen einfachen Mechanismus.)
- Ersetze die einfache Bewertungs-Funktion (TAB[7] – TAB[14]) durch das nun trainierte neuronale Netz
- Heraus kommen sollte dabei eine deulich spiel-stärkere Version des bisherigen Algorithmus

Zunächst wird eine Liste mit realistischen Spiel-Situationen erzeugt, indem man zwei Zufalls-Zahlen-Generatoren gegeneinander spielen läßt



(daß die nicht besonders gut spielen, ist erst einmal egal)

Im zweiten Schritt wird jede Spielsituation konventionell bewertet



Danach wird ein neuronales Netz darauf trainiert, die Bewertungen der Liste möglichst gut zu imitieren.

Nach dem Training wird die statische Bewertungsfunktion des Baumsuch-Algorithmus durch das neuronale Netz ersetzt

Und das Resultat der Aktion spielt dann gar nicht mal so schlecht.....

Einige Passagen aus der Implementation

```
[0]    Z←KALA_START
[1]    A-------------------------------------------------------
[2]    A RH,07.06.2017 Startkonfiguration des KALAHA-Spiels
[3]    A-------------------------------------------------------
[4]    Z←0 (14⍴(6 6 6 6 6 6 0))
[5]
```

```
[0]    Z←KALA_INVERT KONFIG;SPIELER;TAB
[1]    A-------------------------------------------------------
[2]    A RH,12.02.2018 Rollen-Wechsel der Spieler
[3]    A Die Spielsituation wird hier "invertiert"
[4]    A Spieler 0 und Spieler 1 tauschen die Rollen
[5]    A-------------------------------------------------------
[6]    (SPIELER TAB)←KONFIG
[7]    TAB←7⌽TAB
[8]    :if (SPIELER=1)
[9]        SPIELER←0
[10]   :elseif (SPIELER=0)
[11]       SPIELER←1
[12]   :endif
[13]
[14]   Z←(SPIELER TAB)
```

```
[0]    Z←KALA_GAME_OVER KONFIG;SPIELER;TAB;GEW1;GEW2
[1]    A-------------------------------------------------------
[2]    A RH,11.06.2017 Prüfen, ob das Spiel zuende ist
[3]    A
[4]    A-------------------------------------------------------
[5]    (SPIELER TAB)←KONFIG
[6]    Z←(∧/0=TAB[1 2 3 4 5 6])∨(∧/0=TAB[8 9 10 11 12 13])∨(SPIELER=2)
```

Die rekursiv definierte Bewertungsfunktion

```
[0]   Z←TIEFE KALA_BEWERTUNG KONFIG;SPIELER;TAB;GEW;LISTE_ZUEGE;
[1]   ⍝-------------------------------------------------------
[2]   ⍝ RH,03.03.2018 Eine Konfiguration bewerten
[3]   ⍝-------------------------------------------------------
[4]   (SPIELER TAB)←KONFIG
[5]
[6]   ⍝-------------------------------------------------------
[7]   ⍝ Bewertung bei Spiel-Ende
[8]   ⍝ In diesem Fall ist die Bwewertung trivial
[9]   ⍝-------------------------------------------------------
[10]  :if (∧/0=TAB[⍳6])∨(∧/0=TAB[7+⍳6])
[11]      Z←(+/TAB[⍳7])-(+/TAB[7+⍳7])
[12]      →0
[13]  :endif
[14]
[15]  ⍝-------------------------------------------------------
[16]  ⍝ Wenn als Evaluations-Tiefe Null angegeben ist,
[17]  ⍝ wird die Konfiguration nur noch STATISCH bewertet -
[18]  ⍝ d.h. mit einem nicht rekursiven "einfachen"
[19]  ⍝ Algorithmus
[20]  ⍝-------------------------------------------------------
[21]  :if (TIEFE=0)
[22]      Z←KALA_STATIC_BEWERTUNG KONFIG
[23]      →0
[24]  :endif
[25]
[26]  ⍝-------------------------------------------------------
[27]  ⍝ Liste der Nachfolge-Konfigurationen erzeugen
[28]  ⍝ Wichtig dabei ist, daß dabei auch alle Konfigurationen
[29]  ⍝ berücksichtigt sind, die durch Mehrfach-Ziehen
[30]  ⍝ entstehen könnten.
[31]  ⍝-------------------------------------------------------
[32]  LISTE_NACHFOLG←KALA_ALLE_NACHF KONFIG
[33]  LISTE_BEW←(TIEFE-1)⍤KALA_BEWERTUNG⍤¯1⊢LISTE_NACHFOLG
[34]
[35]  ⍝-------------------------------------------------------
[36]  ⍝ Ist der Spieler 1 am Zug, dann wird er sicherlich
[37]  ⍝ die Alternative mit der geringsten Bewertung wählen.
[38]  ⍝ Umgekehrt wird Spieler 0 die Alternative mit der
[39]  ⍝ höchsten Bewertung wählen.
[40]  ⍝-------------------------------------------------------
[41]  :if (SPIELER=1)
[42]      Z←⌊/∈LISTE_BEW
[43]  :elseif (SPIELER=0)
[44]      Z←⌈/∈LISTE_BEW
[45]  :endif
[46]  →0
```

Die statische Bewertungsfunktion

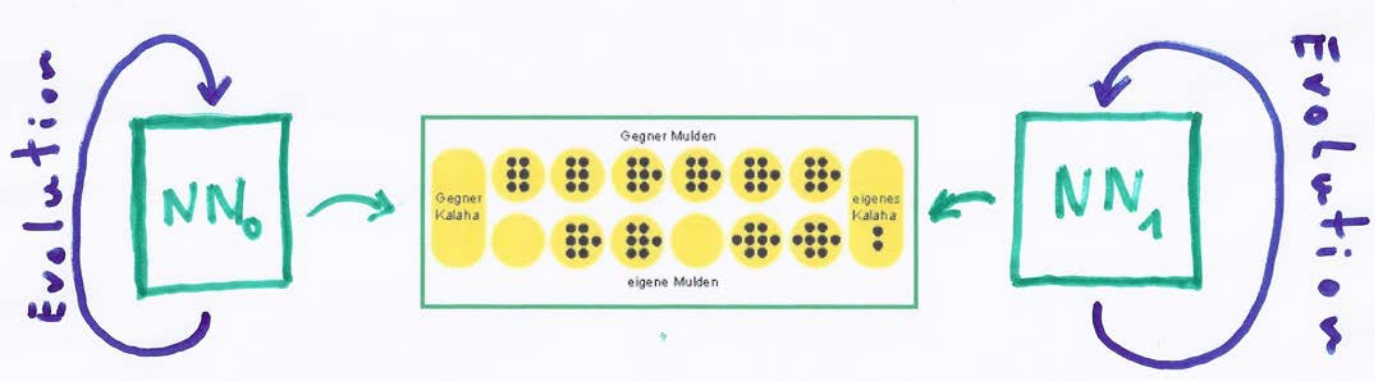
```
[0]   Z←KALA_NN_STATIC_BEWERTUNG RA;NN;KONFIG;SPIELER;TAB;REVER
[1]   A-------------------------------------------------------
[2]   A RH.03.03.2018  Statische Bewertung
[3]   A                mit Hilfe eines neuronalen Netzes
[4]   A
[5]   A-------------------------------------------------------
[6]   (NN KONFIG)←RA
[7]   (SPIELER TAB)←KONFIG
[8]   REVERS_BEW←(SPIELER=1)
[9]
[10]  A-------------------------------------------------------
[11]  A Da das Netz nur auf Situationen trainiert ist,
[12]  A bei denen Spieler0 am Zug ist, muß die Konfiguration
[13]  A evtl. zuvor invertiert werden
[14]  A-------------------------------------------------------
[15]  :if REVERS_BEW
[16]      KONFIG←KALA_INVERT KONFIG
[17]      (SPIELER TAB)←KONFIG
[18]  :endif
[19]
[20]  A---Die Kalahas zuerst leeren------------
[21]  (K0 K1)←TAB[7 14]
[22]  INP←TAB[(⍳14)~(7 14)]
[23]
[24]  A---Dann die Bewertung durchführen-------
[25]  WERT←10×+/KALA_NN_CALCULATE NN INP
[26]  WERT←WERT+K0-K1
[27]
[28]  :if REVERS_BEW
[29]      WERT←-WERT
[30]  :endif
[31]  Z←WERT
[32]
[33]  →0
```

Die Misserfolge

Was gar nicht funktioniert hat, waren Versuche, neuronale Netze unmittelbar (d.h. ganz ohne Baumsuch-Algorithmus) für das Spiel zu trainieren.

Die Spiel-Stärken der Netze waren nach dem Training in allen Fällen völlig unbefriedigend ....

Misserfolg 1
Co-Evolution zweier neuronaler Netze

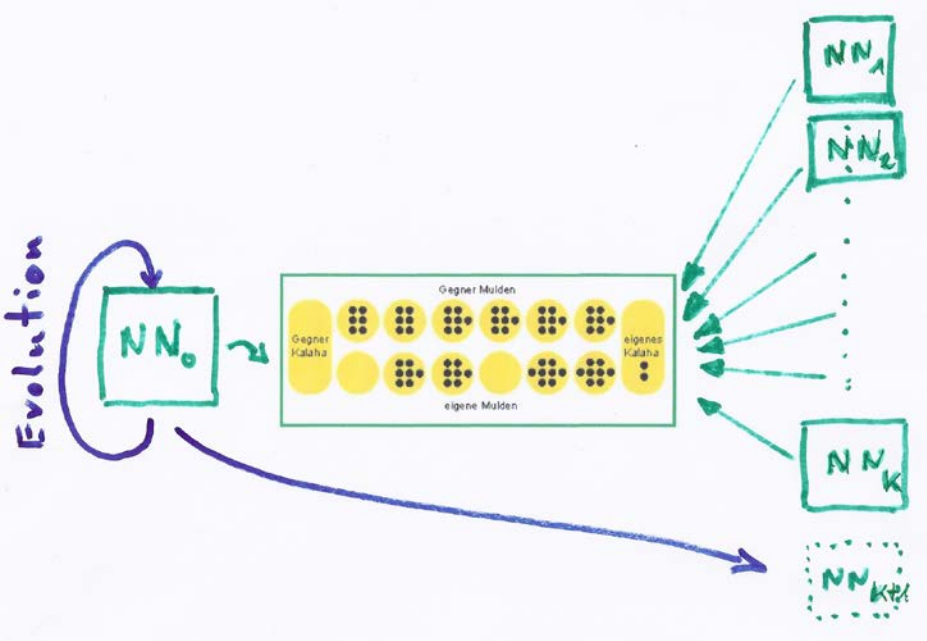


Die Idee dahinter:

Man läßt zwei neuronale Netze sehr oft gegeneinander spielen (bzw. ein Netz gegen sich selbst) und versucht, die Spiel-Stärke durch eine Art Evolutionsprozess immer weiter zu steigern.

Hört sich gut an – funktioniert aber nicht

Misserfolg 2
Verschärfte Evolutions-Bedingungen

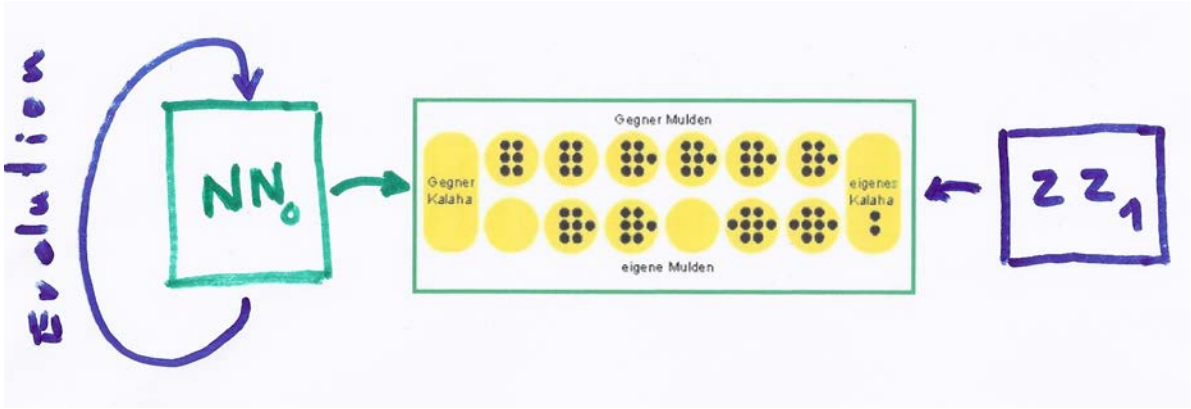


Die Idee dahinter:

Man läßt ein Netz NN_0 gegen eine ganze Liste von Gegnern spielen und etabliert einen entsprechenden Evolutionsmechanismus. Wenn NN_0 alle seine Gegner schlagen kann, kommt es selbst auf die Liste und das Spiel geht mit einem neuen „Herausforderer“ von neuem los ....

(funktioniert aber auch nicht)

Misserfolg 3
Spiel gegen einen Zufallszahlen-Generator

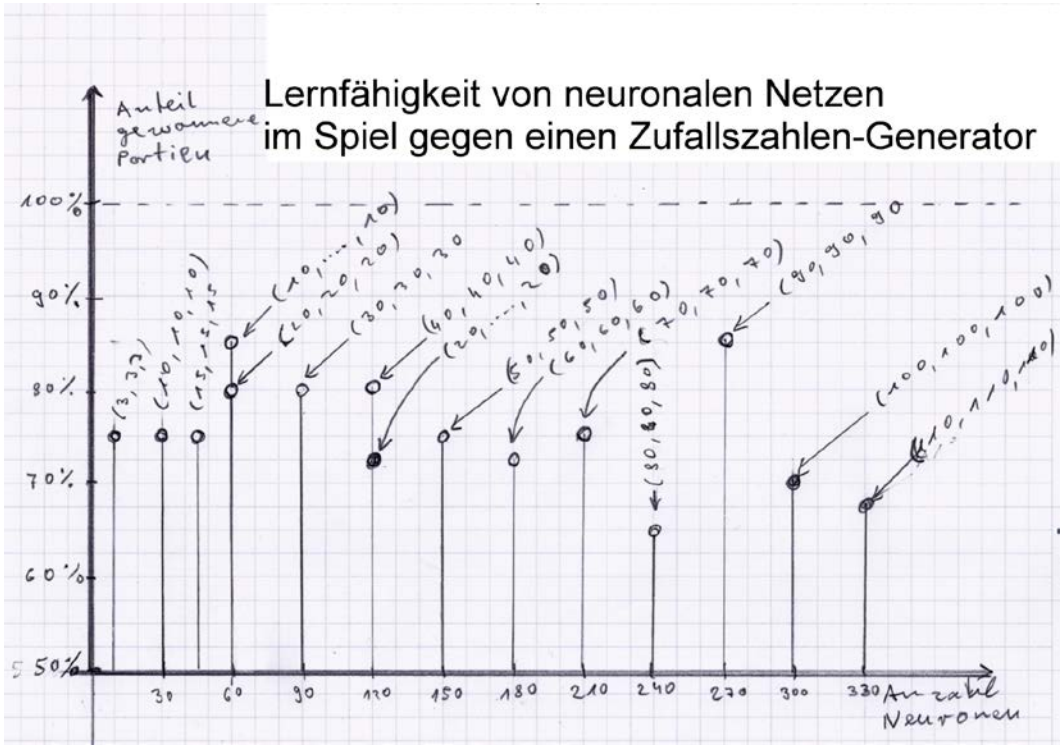


Die Idee dahinter:

NN_0 wird darauf trainiert, gegen einen Zufallszahlen-Generator eine möglichst hohe Quote von Spielen zu gewinnen. Wenn NN_0 zuverlässig gegen einen Zufallszahlen-Generator gewinnen kann, gewinnt es evtl. auch gegen einen menschlichen Gegner .....

(funktioniert aber auch nicht)

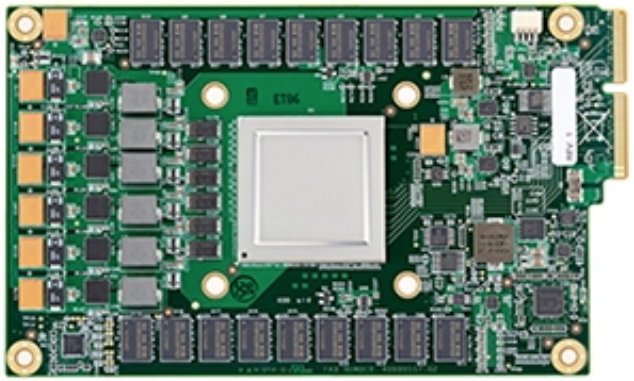
Wenigstens eine Erkenntnis:
Es kommt nicht auf die Größe an

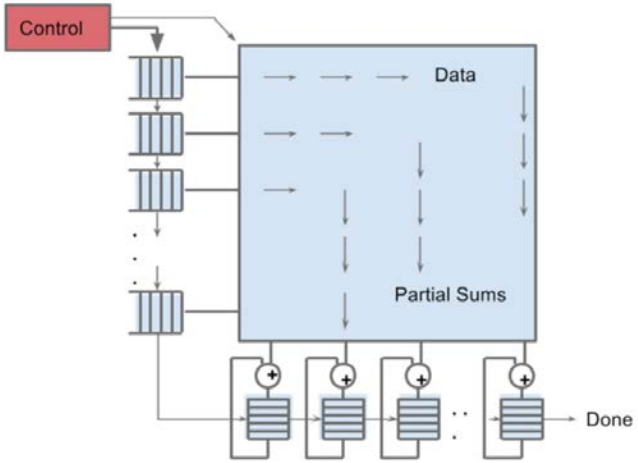


Bei den Spielen gegen einen Zufallszahlen-Generator erreichte keines der getesteten Netze eine Gewinn-Quote von mehr als 85% . Die erreichten Spielstärken waren weitgehend unabhängig von der Anzahl der verwendeten Neuronen.

Ergo: Es ist nicht so einfach, wie es sich in den Veröffentlichungen immer anhört ....

Noch ein kurzer Blick auf GOOGLEs „Tensor-Processing-Unit“



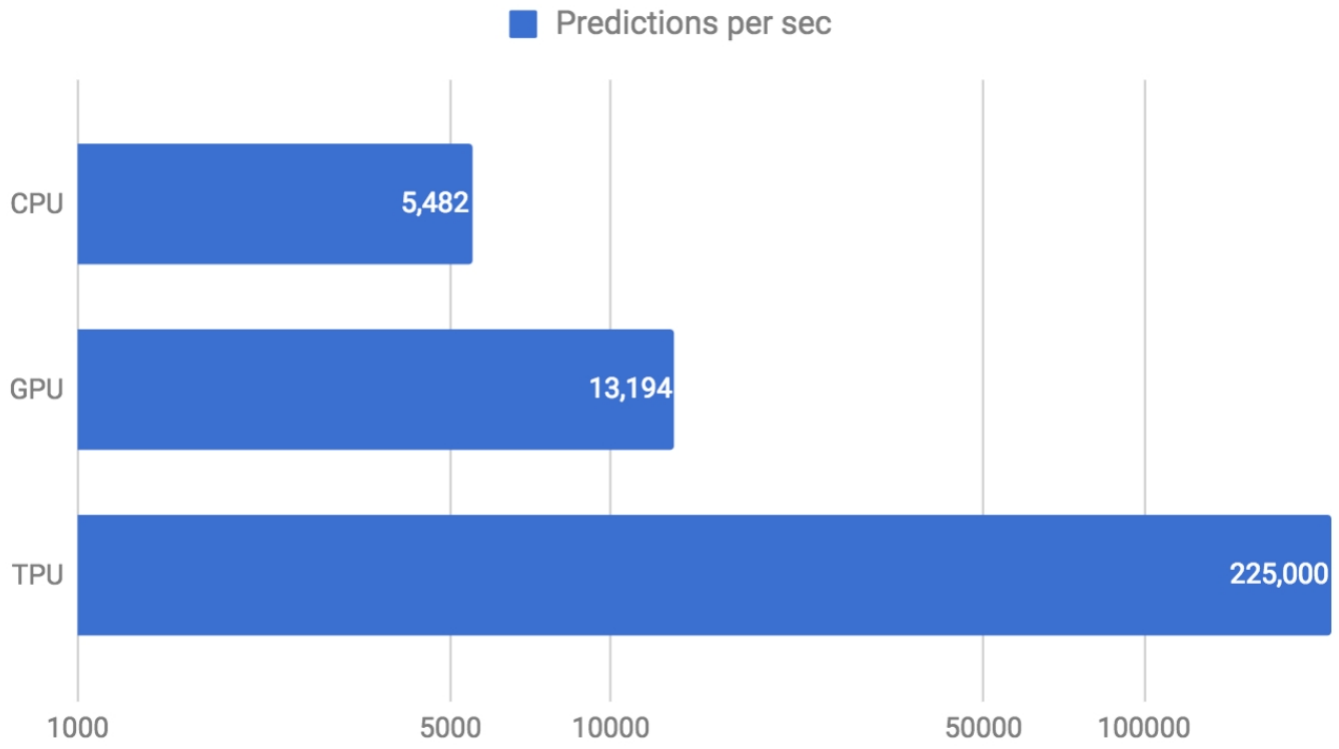


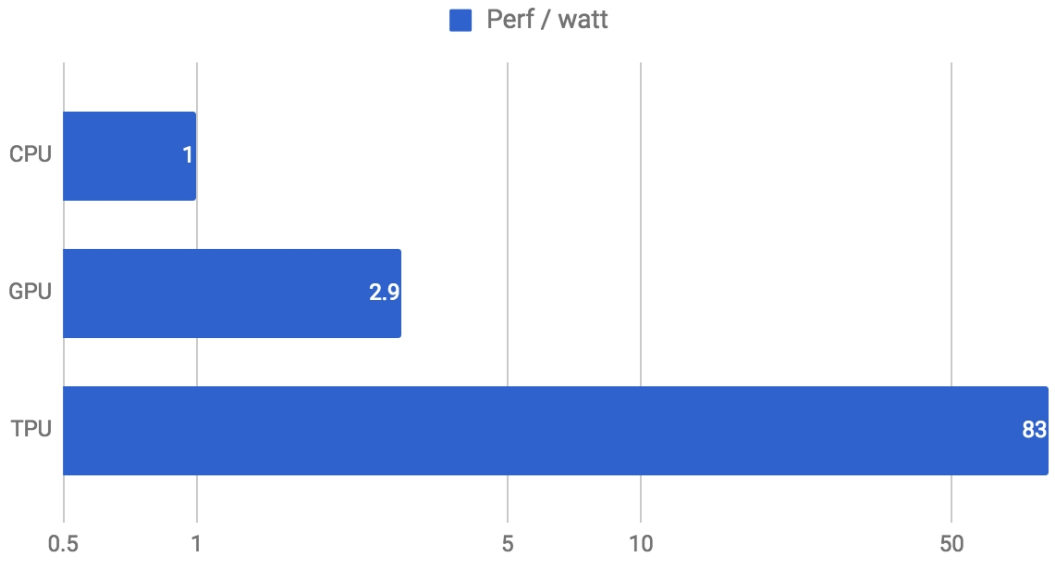
Dieser Single-Purpose-Prozessor wurde speziell für die Anwendung in neuronalen Netzen entwickelt und realisiert im wesentlichen lediglich eine **8-Bit-Matrix-Multiplikation**.

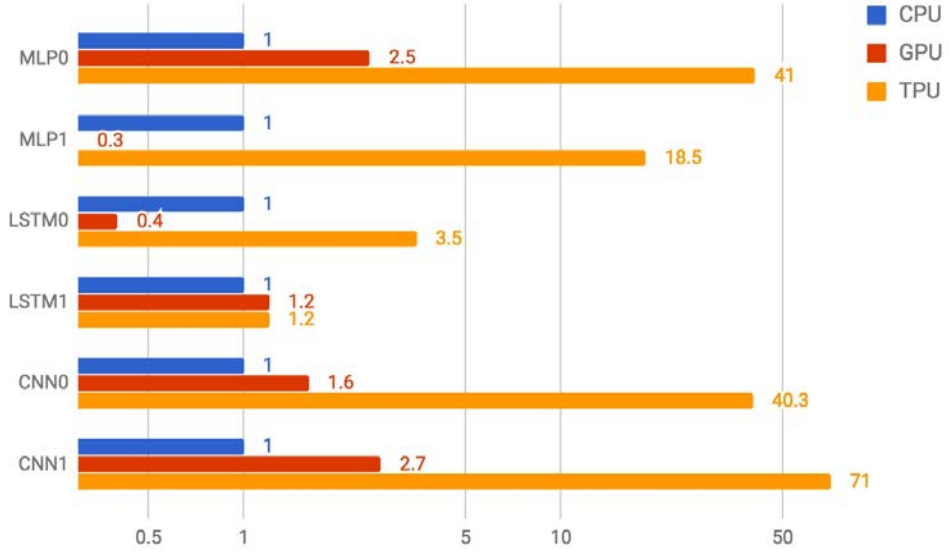
Aber: Die Matrizen sind sehr groß und werden parallel abgearbeitet

Die stark eingeschränkte Funktionalität ermöglicht eine relativ kurze Entwicklungszeit und eine hohe Effizienz beim Betrieb des Prozessors.

Die Performance-Werte sind beeindruckend (wenn man der GOOGLE-Werbung glauben will)

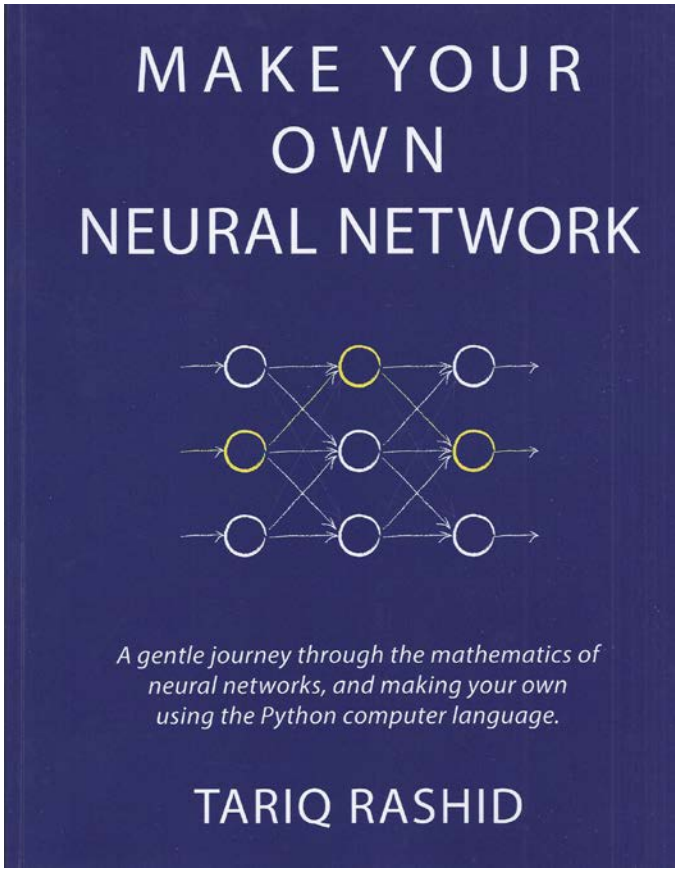



Perf / watt
CPU
GPU
TPU
1
2.9
83
0.5
1
5
10
50


CPU
GPU
TPU
MLP0
MLP1
LSTM0
LSTM1
CNN0
CNN1
1
2.5
41
1
0.3
18.5
1
0.4
3.5
1
1.2
1.2
1
1.6
40.3
1
2.7
71
0.5
1
5
10
50

Literatur-Empfehlung

für alle, die in das Thema einmal „hineinschnuppern“ wollen



Ausblick

Neuronale Netze gibt es bereits seit Mitte der 60-er Jahre, aber außer einigen Nischen-Anwendungen hatten sie bisher keine nennenswerte Bedeutung, da die notwendige Rechnerkapazität in der Regel nicht verfügbar war.

Mit der Nutzung von **GPUs** als billiger Alternative zu Super-Rechnern hat sich dies in den vergangenen 10 Jahren drastisch geändert. Viele Anwendungen lassen sich nun mit neuronalen Netzen realisieren, die zuvor unrealistisch teuer gewesen wären.

Mit der Nutzung von **TPUs** als billiger Alternative zu **GPUs** ist GOOGLE jetzt noch einen Schritt weiter gegangen und hat einen Prozessor-Typ vorgestellt, der allein für die Durchführung von Matrix-Operationen konstruiert wurde.

Weitere Verbesserungen der Performance sind zu erwarten .....

Fazit

Wie in neuronales Netz funktioniert, weiß man im Prinzip seit langer Zeit.

Wie man ein solches Netz am sinnvollsten trainieren kann, ist aber im Moment wohl noch eher eine Kunst als eine Wissenschaft.

Neue leistungsfähigere Typen von neuronalen Netzen sind in den verangenen Jahrzehnten entwickelt worden. Sie können z.T. Aufgaben lösen, für die die Perceptrons der 60-er Jahre nicht geeignet waren.

Mit der Verfügbarkeit kostengünstiger Hardware wird sich die Bandbreite von Anwendungen, die auf der Technik neuronaler Netze basieren, deutlich vergrößern.

Zu welchen Leistungen diese Mechanismen dann in der Lage sein werden, ist heute noch ziemlich unklar. Eine Reihe von spektakulären Erfolgen läßt aber Einiges erwarten.

Mittelfristig wird man nicht darum herum kommen, sich mit dieser Technologie auseinanderzusetzen.

Es bleibt spannend ....