Skript zur Vorlesung
Künstliche Neuronale Netze (Deep Learning)

im SoSe 2022
an der Technischen Hochschule Mittelhessen

Andreas Dominik

4. April 2022

Inhaltsverzeichnis

Kapitel 1:

Biologische Grundlagen neuronaler Netze

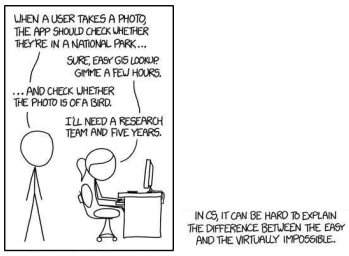
1.1 Warum DNNs

In der Informatik scheinen manche Aufgeben leicht und erweisen sich als unlösbar:

In the 60s, Marvin Minsky assigned a couple of undergrads to spend the summer programming a computer to use a camera to identify objects in a scene.

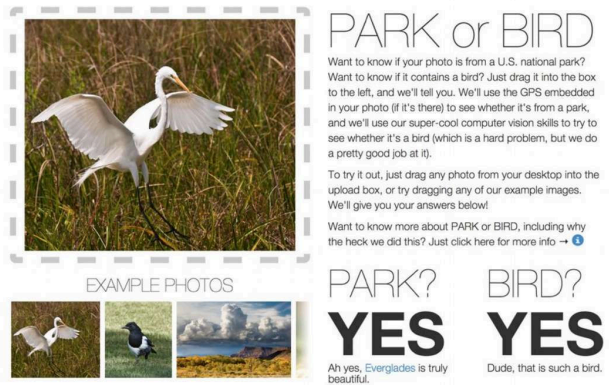
He figured they'd have the problem solved by the end of the summer.

Half a century later, we're still working on it ... or not!



Erst Jahrzehnte später kann das Problem leicht gelöst werden - mit Hilfe künstlicher neuronaler Netze (die es übrigens 1970 auch schon gab!)

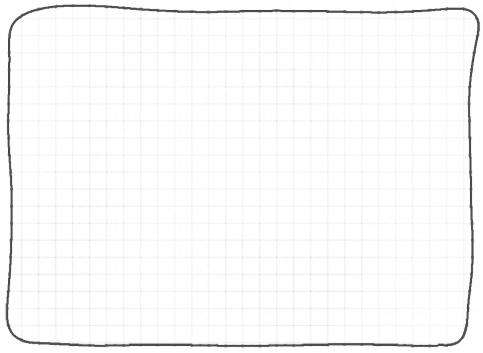
http://code.flickr.net/2014/10/20/introducing-flickr-park-or-bird/



Wa sit der Unterschied?

- 2 + 2 =
- 5 * 125 =
- 234567 / 654 =

Eigenschaft	Computer	Gehirn
Prozessoren	1 (10^9 Transistoren)	10^{12} Neurone
Komplexität einer CPU	groß	minimal
Komplexität der Vernetzung	klein	groß
Taktung	sequenziell	parallel
Kommunikation	minimiert	maximiert
Taktfrequenz	groß (schnell); 10^9Hz	klein (langsam); 100Hz
Präzision	hoch (genau)	klein (fuzzy)
Speicherung	an einem Ort	verteilt
Programm	designed	gelernt
Speicherung des Programms	Software	Hardware
Energieverbrauch	30 Watt	30 Watt



1.2 Wozu haben wir ein Gehirn?

Schachspielen ist einfacher als gedacht!

Ein Bier einschenken ist schwieriger als gedacht!

Hochschule Weingarten

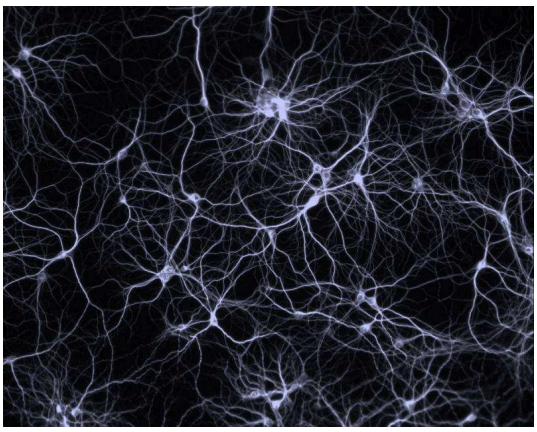
Tischtennis bis heute unmöglich!

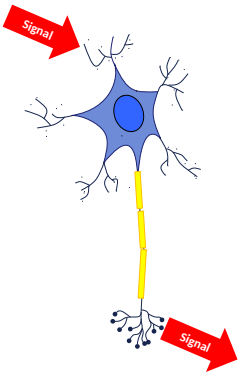
Kuka vs. Timo Boll

Wer festsitzt braucht kein Gehirn!

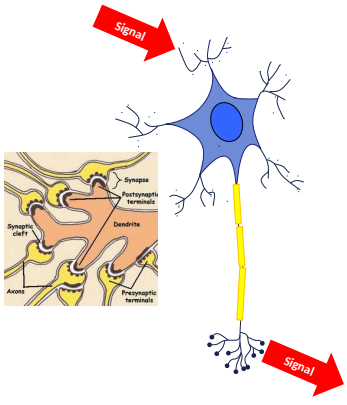
Bewegung ist der Schlüssel zum Training des neuronalen Netzes:

1.3 Biologische Nervenzellen

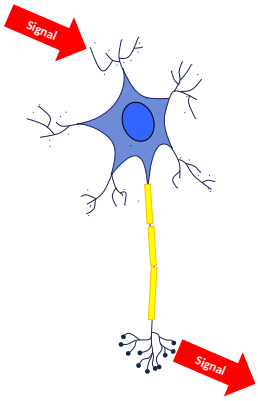




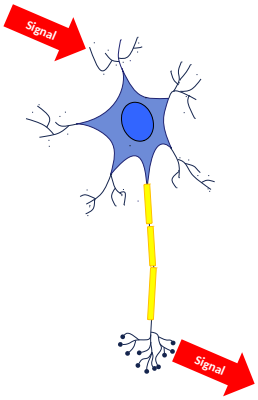
Dentrit:

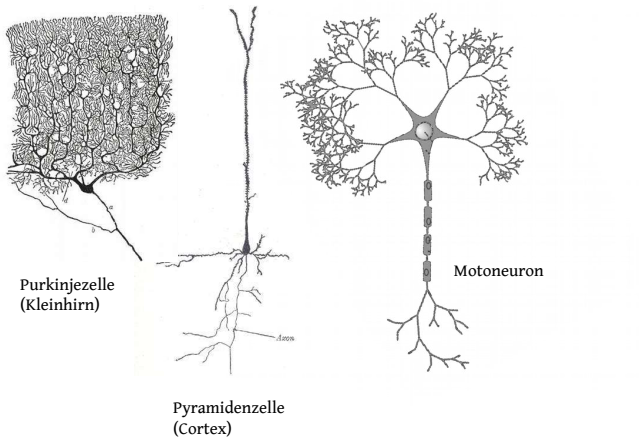


Zellkörper:



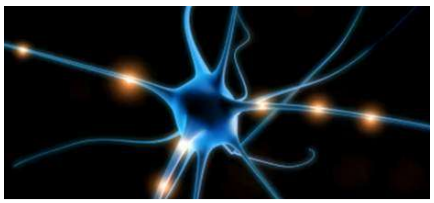
Axon





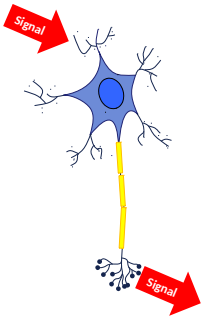
Die Evoultion des Gehirns:

The origin of the brain:
http://www.youtube.com/watch?v=6RbPQG9WTZM

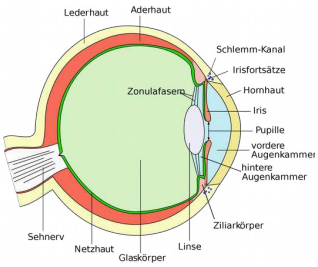


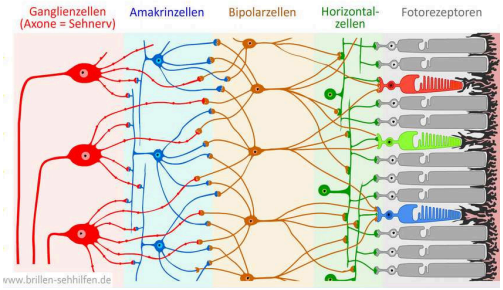
Informationsübertrgung in Salven:

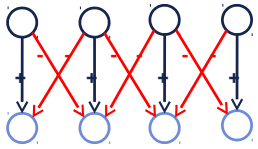
- Binäres Signal
- Frequenzmodulation
- Verzögerung

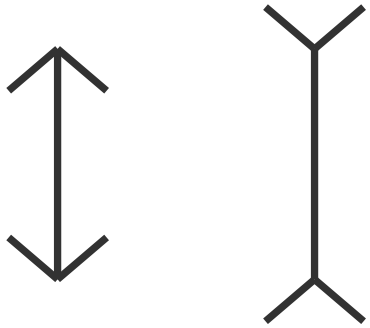


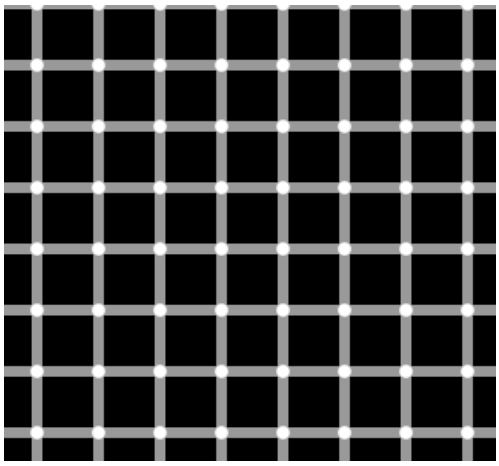
1.4 Neuronale Verschaltung im Auge

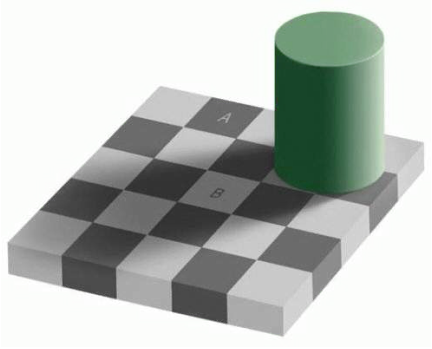


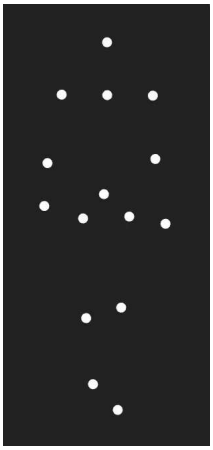




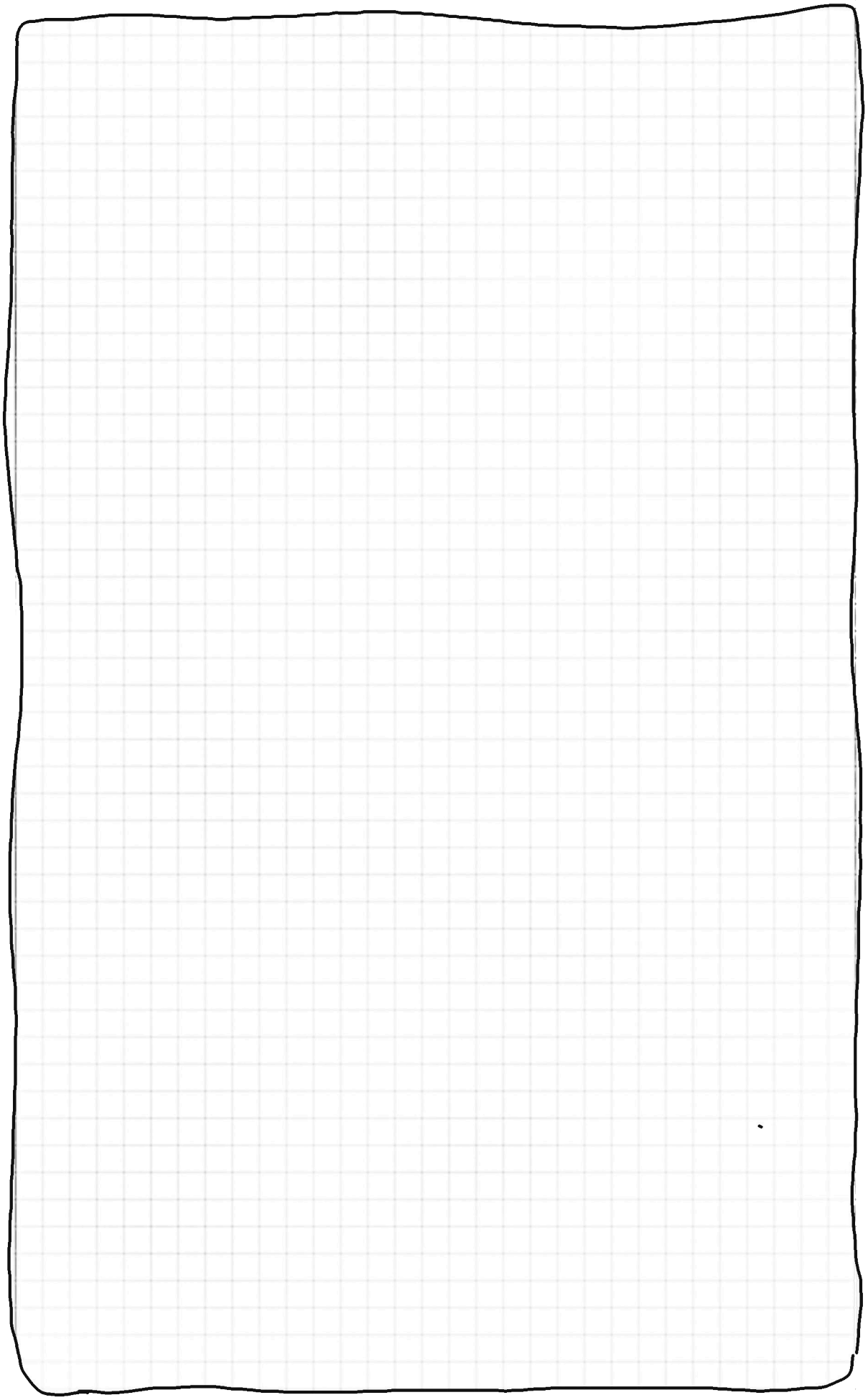








http://www.biomotionlab.ca/Demos/webgl_walker/webgl_walker.php
G. Johansson, „*Visual perception of biological motion and a model for its analysis*“, Percept. Psychophys., **14** (1973) 201–211.

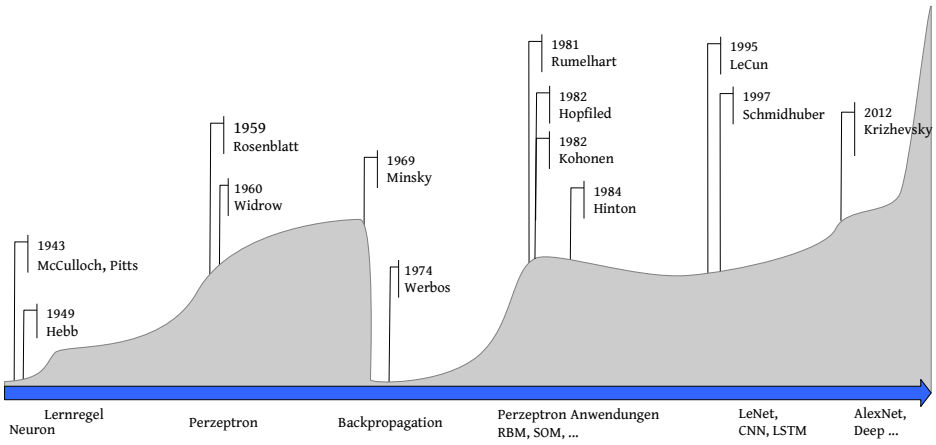


Kapitel 2:

Eine kurze Geschichte der künstlichen neuronalen Netze

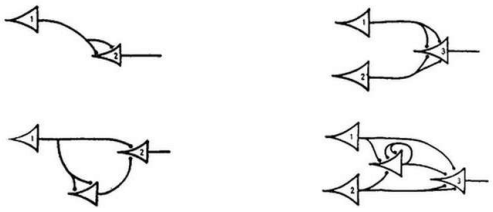
2.1 Die Anfänge

Hypes and Stagnations

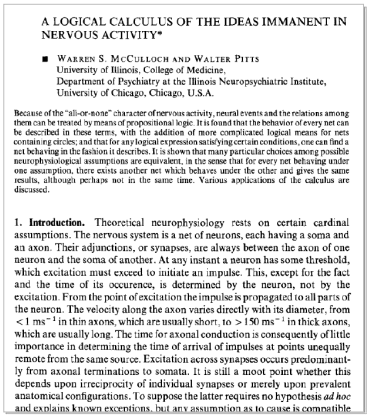


1943: Warren McCulloch und Walter Pitts

- Neurone
- Aktivierende und hemmende Verbindungen
- Wo kommen die Verbindungen her?



W.S. McCulloch, W. Pitts; A logical calculus for neural activity; *Bull. Math. Biophys.* **5** (1943) 115-133.

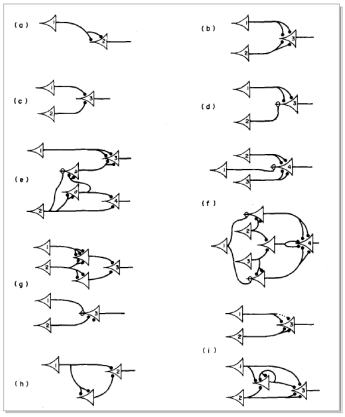


A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY*

■ WARREN S. MCCULLOCH AND WALTER PITTS
University of Illinois, College of Medicine,
Department of Psychiatry at the Illinois Neuropsychiatric Institute,
University of Chicago, Chicago, U.S.A.

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

1. **Introduction.** Theoretical neurophysiology rests on certain cardinal assumptions. The nervous system is a net of neurons, each having a soma and an axon. Their adjunctions, or synapses, are always between the axon of one neuron and the soma of another. At any instant a neuron has some threshold, which excitation must exceed to initiate an impulse. This, except for the fact and the time of its occurence, is determined by the neuron, not by the excitation. From the point of excitation the impulse is propagated to all parts of the neuron. The velocity along the axon varies directly with its diameter, from $< 1\ ms^{-1}$ in thin axons, which are usually short, to $> 150\ ms^{-1}$ in thick axons, which are usually long. The time for axonal conduction is consequently of little importance in determining the time of arrival of impulses at points unequally remote from the same source. Excitation across synapses occurs predominantly from axonal terminations to somata. It is still a moot point whether this depends upon irreciprocity of individual synapses or merely upon prevalent anatomical configurations. To suppose the latter requires no hypothesis *ad hoc* and explains known exceptions, but any assumption as to cause is compatible



W.S. McCulloch, W. Pitts; A logical calculus for neural activity; *Bull. Math. Biophys.* **5** (1943) 115-133.

Eigenschaften:

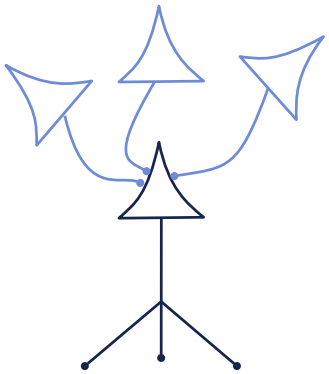
-
-
-
-

Ergebnisse:

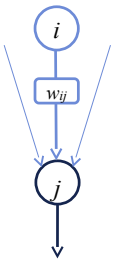
-
-

Modifikation:

-



Parameter des McCulloch-Pitts-Neurons



a_i Aktivierungszustand — 0 oder 1

$o_i = f(a_i)$ Output — Identität

$w_{i,j}$ Gewichte

$net_j = f(o_{i..n}, w_{i..n,j}) = \sum_i o_i \cdot w_{ij}$ Netzeingabe

θ_j Schwellenwert *(Theta)*

$a_j(t+1) = f_{act}(a_j(t), net_j(t), \theta_j)$ Schwellenwertfunktion

Was ist möglich:

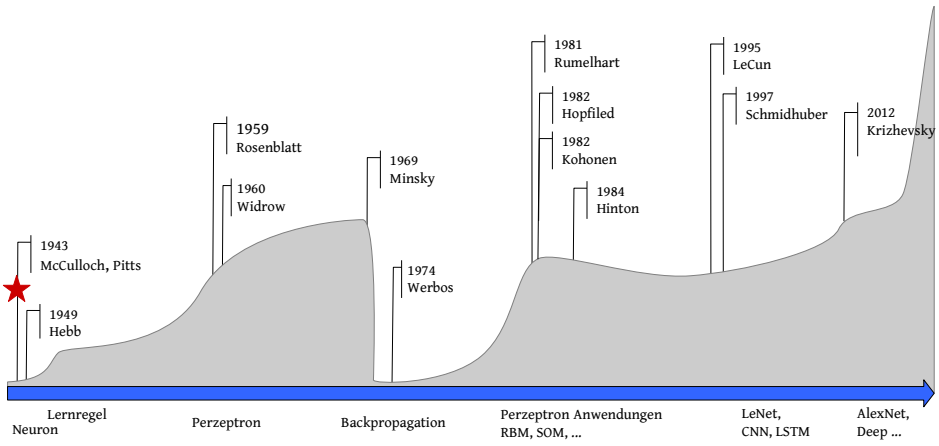
- Ein neuronales Netz läßt sich aus einfachen McCulloch-Pitts-Neuronen aufbauen
- Erstaunliche Leistung möglich:
  - Logische Gatter
  - Klassifikatoren für linear separierbare Probleme.

Was nicht:

- Vernetzung und Gewichte müssen genau überlegt werden
- Keine Selbstorganisation
- Kein Lernen.

2.2 Donald Olding Hebb: Lernregel

Hypes and Stagnations



1949: Donald Olding Hebb

Pawlows Hund *Baikal*; Pawlow Museum Ryazan, Russland.

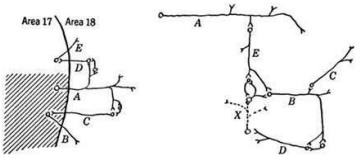
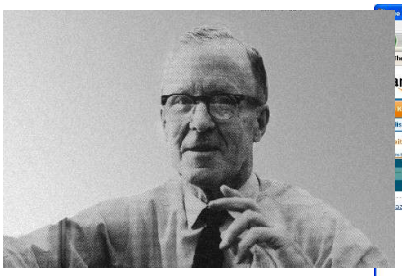


FIGURE 8 FIGURE 9

FIGURE 8. Cells *A* and *B* lie in a region of area 17 (shown by hatching) which is massively excited by an afferent stimulation. *C* is a cell in area 18 which leads back into 17. *E* is in area 17 but lies outside the region of activity. See text.

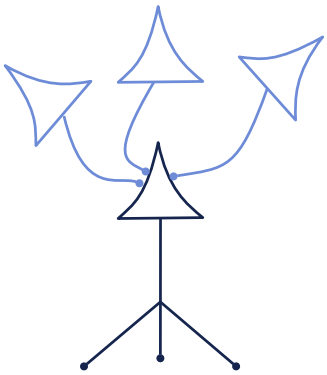
FIGURE 9. *A*, *B*, and *C* are cells in area 18 which are excited by converging fibers (not shown) leading from a specific pattern of activity in area 17. *D*, *E*, and *X* are, *among the many cells with which* A, B, *and* C *have connections*, ones which would contribute to an integration of their activity. See text.

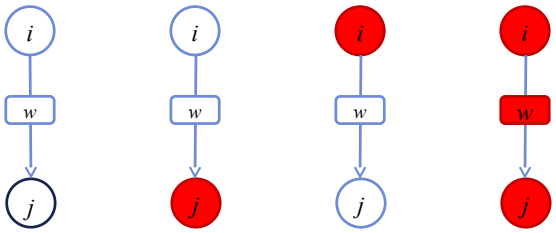
D.O. Hebb; *The Organization of Behavior: A Neuropsychological Theory* (1949) Wiley & Sons, New York.

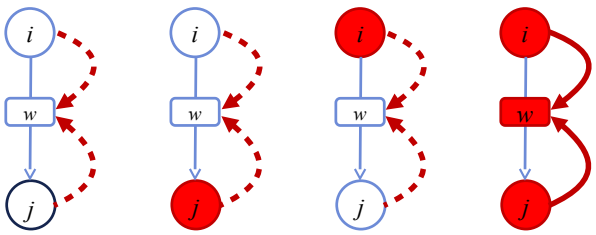


Hebbsche Regel

„Wenn ein Axon der Zelle A [. . .] Zelle B erregt und wiederholt und dauerhaft zur Erzeugung von Aktionspotenzialen in Zelle B beiträgt, so resultiert dies in Wachstumsprozessen oder metabolischen Veränderungen in einer oder in beiden Zellen, die bewirken, dass die Effizienz von Zelle A in Bezug auf die Erzeugung eines Aktionspotenzials in B größer wird."







Was ist möglich:

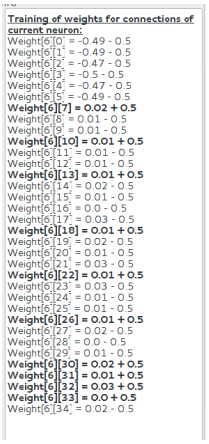
- Ein neuronales Netz läßt sich aus einfachen McCulloch-Pitts-Neuronen aufbauen
- Das Netz organisiert sich selbst.

Fragen:

- Kann das Netz lernen?
- Kann das Netz Erregungen verarbeiten, Muster speichern und wieder abrufen?

Hopfield-Simulator nach der Hebbschen Regel: Neuronicus

- https://homepages.thm.de/~admn12/pages/research/projectNeuronicus.html
- Simulator mit erweiterter Hebb-Regel:
  „Lerne, wenn Quell- und Zielneuron gleich aktiviert sind!“



Assoziativer Speicher:

- Trainieren Sie 1 Muster!
- Trainieren Sie 2-3 Muster!

Was ist möglich:

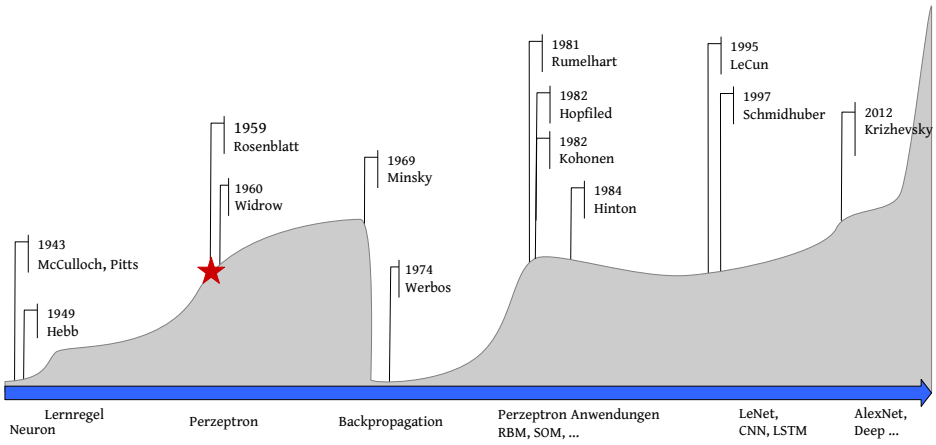
- Ein neuronales Netz läßt sich aus einfachen McCulloch-Pitts-Neuronen aufbauen
- Das Netz organisiert sich selbst
- Das Netz kann lernen
- Das Netz kann Muster speichern und wieder abrufen.

Fragen:

- Wie kann man so etwas 1950 bauen?

2.3 Frank Rosenblatt: Erstes Perceptron

Hypes and Stagnations



1958: Frank Rosenblatt: Mark I Perceptron

- Eingabematrix: 20x20
- 512 gewichtete Verbindungen

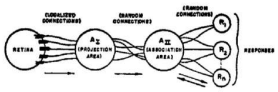
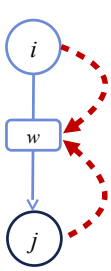
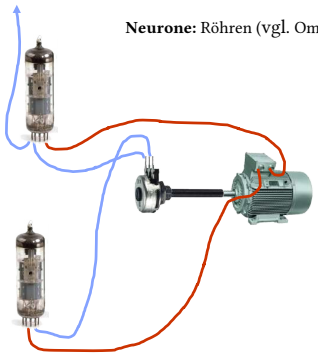


FIG. 1. Organization of a perceptron.

F. Rosenblatt, *„The perceptron: a probabilistic model for information storage and organization in the brain"*, Physiol. Rev., **65** (1958) 386 – 408.





Neurone: Röhren (vgl. Omas Radio)

Gewichtete Axone: Drehpotenziometer (vgl. Dimmer)

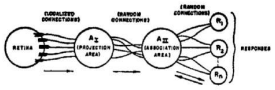
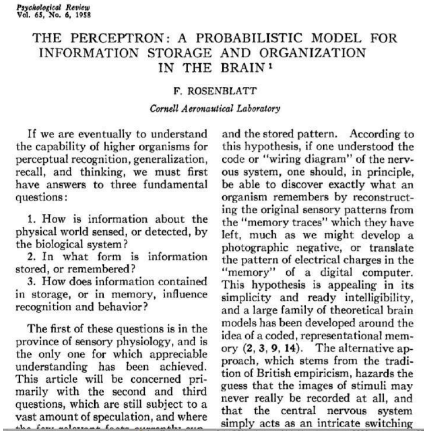


FIG. 1. Organization of a perceptron.

F. Rosenblatt, *„The perceptron: a probabilistic model for information storage and organization in the brain.“*, Physiol. Rev., Bd. 65, Nr. 6, S. 386 – 408, 1958.



Psychological Review
Vol. 65, No. 6, 1958

THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN [1]

F. ROSENBLATT

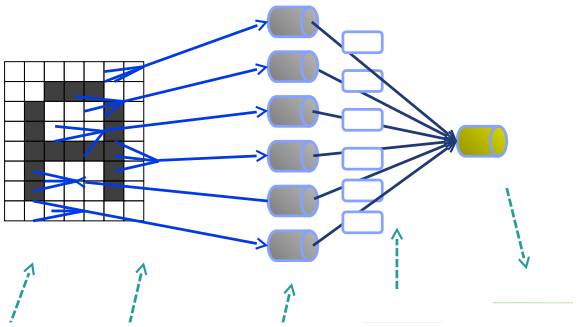
Cornell Aeronautical Laboratory

If we are eventually to understand the capability of higher organisms for perceptual recognition, generalization, recall, and thinking, we must first have answers to three fundamental questions:

1. How is information about the physical world sensed, or detected, by the biological system?
2. In what form is information stored, or remembered?
3. How does information contained in storage, or in memory, influence recognition and behavior?

The first of these questions is in the province of sensory physiology, and is the only one for which appreciable understanding has been achieved. This article will be concerned primarily with the second and third questions, which are still subject to a vast amount of speculation, and where

and the stored pattern. According to this hypothesis, if one understood the code or "wiring diagram" of the nervous system, one should, in principle, be able to discover exactly what an organism remembers by reconstructing the original sensory patterns from the "memory traces" which they have left, much as we might develop a photographic negative, or translate the pattern of electrical charges in the "memory" of a digital computer. This hypothesis is appealing in its simplicity and ready intelligibility, and a large family of theoretical brain models has been developed around the idea of a coded, representational memory (**2**, **3**, **9**, **14**). The alternative approach, which stems from the tradition of British empiricism, hazards the guess that the images of stimuli may never really be recorded at all, and that the central nervous system simply acts as an intricate switching



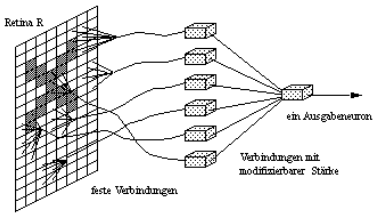
- Eingabeneuronen: Dumm
- Hidden Neurons: 1 Schicht
- 1 Ausgabeneuron, mit jedem Hidden Neuron verbunden

Lernen:

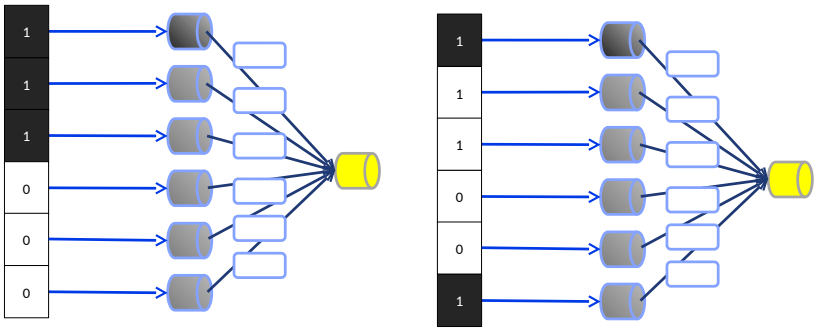
- Nur die Gewichte der zweiten Schicht!
- Hebbsche Regel mit Teaching Input

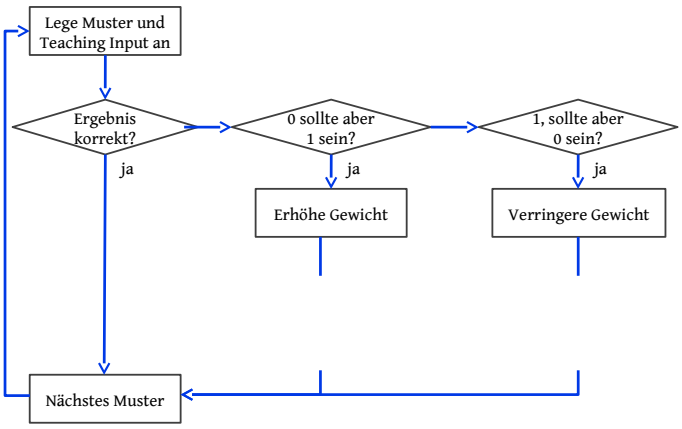
Frage:

- Frage: Wann wird ein Gewicht verändert?



Wie lernt das Mark I Perceptron?

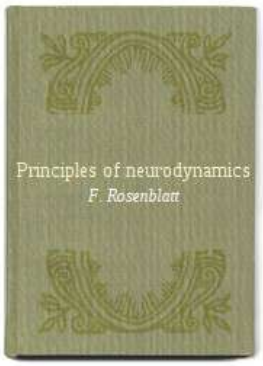




1962, Frank Rosenblatt:

Perzeptron-Konvergenz-Theorem:

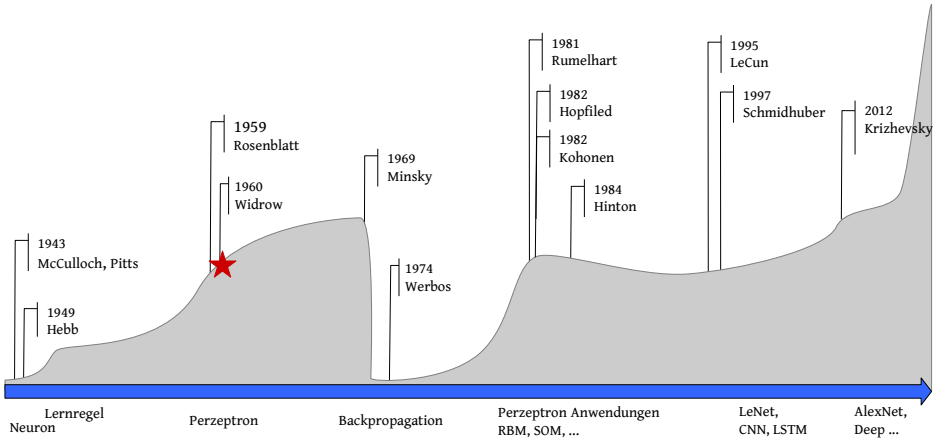
„... Der Lernalgorithmus des Prezeptrons konvergiert in endlicher Zeit..."



Das Perceptron kann in endlicher Zeit alles lernen, was es repräsentieren kann

2.4 Bernard Widrow und Ted Hoff: Deltaregel

Hypes and Stagnations



Deltaregel/Widrow-Hoff-Regel

- Verbesserte Lernregel für das Perzeptron: Korrekturen abhängig vom Fehler
- Memistor Corporation

Marcian Edward „Ted“ Hoff Jr.

- 1968: 12. Mitarbeiter bei Intel: universelle Schaltung!
- 1980 CTO Intel
- 1983 VC Atari

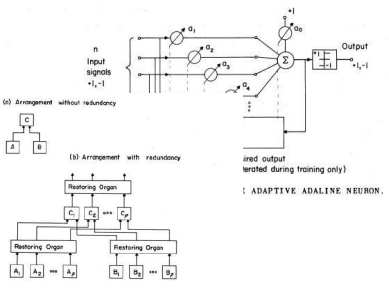


FIG. 2. PLACEMENT OF RESTORING ORGANS IN A REDUNDANT CIRCUIT, ILLUSTRATED FOR A SIMPLE TREE CIRCUIT.

ired output
terated during training only)

: ADAPTIVE ADALINE NEURON.

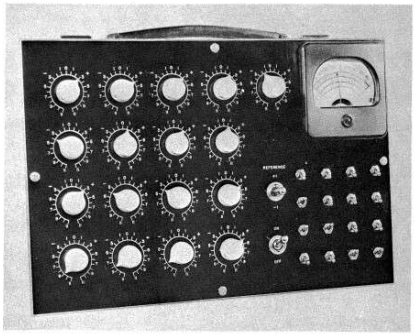
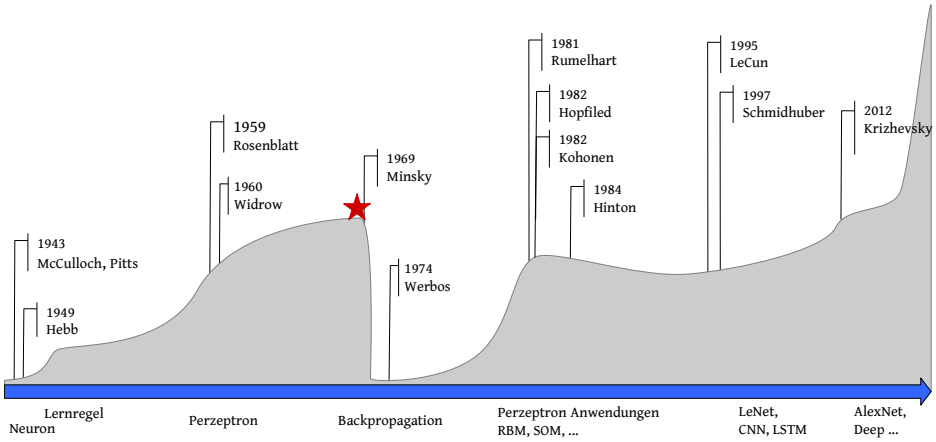


FIG. 9. A MANUALLY-ADAPTED ADALINE NEURON.

2.5 Marvin Minski: der AI-Winter

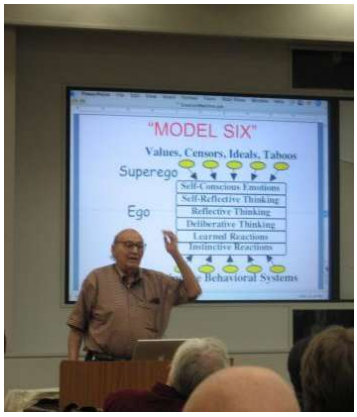
Hypes and Stagnations



Marvin Minski, Seymour Papert

KI-Definition von Minsky:

- „... Aufgaben zu lösen, zu deren Lösung Intelligenz notwendig ist, wenn sie vom Menschen durchgeführt werden ...“



Neuronale netze versus Symbnolische KI

Beispiel Cyc-Projekt:

Datenbank des impliziten Wissens!

ABER:

Das Perzeptron kann nur einfache Probleme repräsentieren (Beispiel. OR aber nicht XOR)

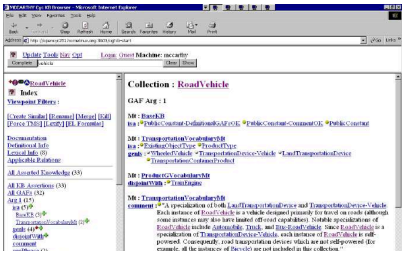
Marvin Minsky:

Artificial neural networks are “Research Dead End.“

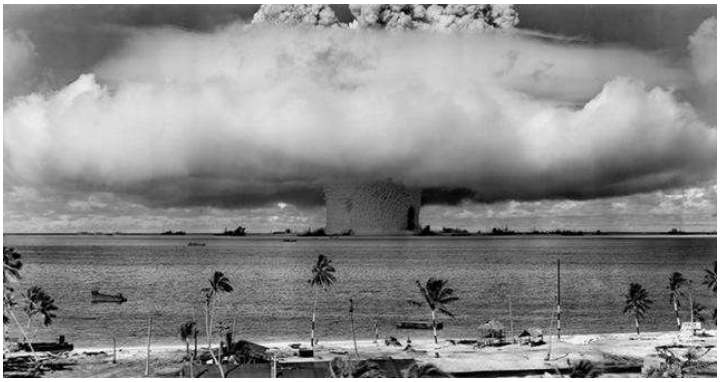
"People have silly reasons why computers don't really think. The answer is we haven't programmed them right; they just don't have much common sense.

There's been only one large project to do something about that, that's the famous Cyc project.“

Marvin Minsky, MIT, May 2001



AI-Winter: 1970s und 1980s



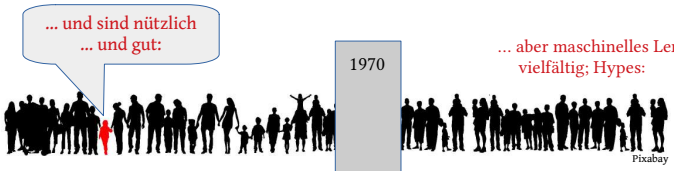
TTAPS:
R. P. Turco, O. B. Toon, T. P. Ackerman, J. B. Pollack, Carl Sagan;
Nuclear Winter: Global Consequences of Multple Nuclear Explosions.
Science **222** (1983) 1283-1292.
Bild: Pexels.

Starke KI

Pixabay

Schwache KI

Novag Robot Adversary (1982)

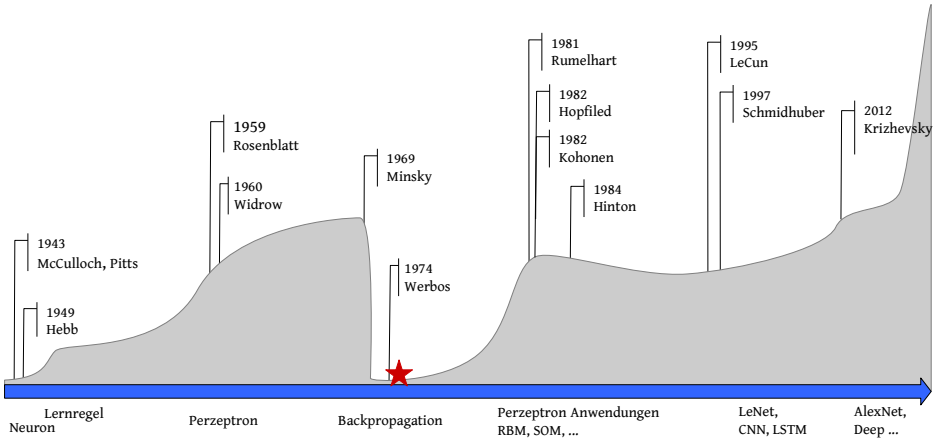


- Backpropagation (Werbos, Rumelhart, Training mehrschichtiger Netze)
- Selbstorganisierende Hopfiled-Netze
- Bolzmann-Maschinen (Simulated Annealing)
- Rekurrente Netze (Jordan und Elman) für zeitabhängige Daten
- CNNs (LeCun) zur Bilderkennung
- LSTMs (Schmidhuber) lösen das *vanishing gradient* Problem.

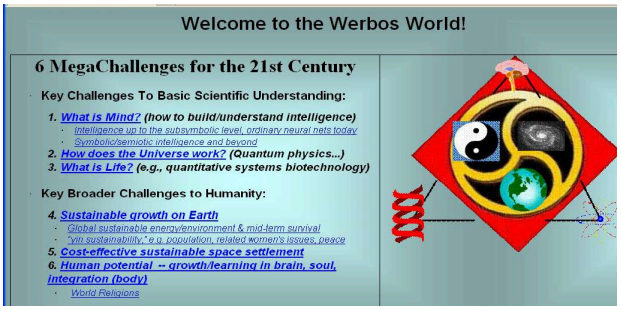
- Regressionsmodelle
- Hidden Markow Modelle
- Support Vector Machines
- Random Forrests
- Boosting

2.6 Paul Werbos: Backpropagation

Hypes and Stagnations

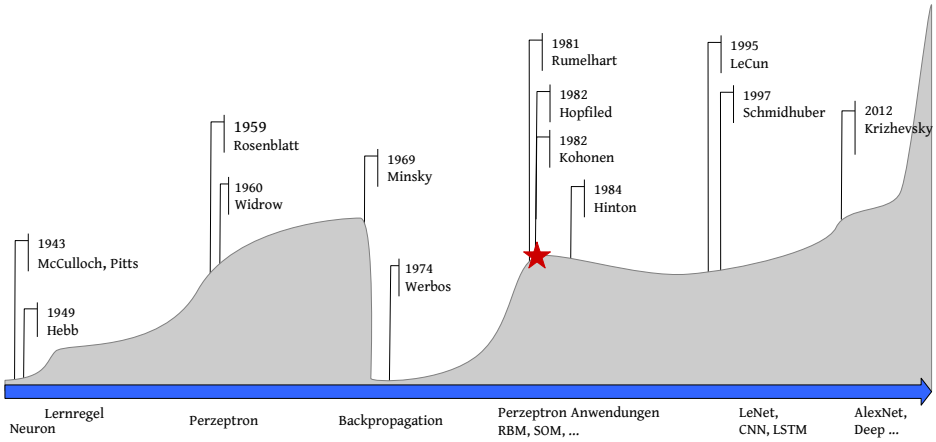


- Paul Werbos entwickelt das Backpropagation Verfahren, zum Training mehrstufiger Netze
- Rumelhardt macht es 10 Jahre später bekannt
- Jetzt können endlich mehrstufige Netze trainiert werden!



2.7 Hopfield und Kohonen: A new hope

Hypes and Stagnations



Proc. Natl. Acad. Sci. USA
Vol. 79, pp. 2554–2558, April 1982
Biophysics

Neural networks and physical systems with emergent collective computational abilities

(associative memory/parallel processing/categorization/content-addressable memory/fail-soft devices)

J. J. HOPFIELD

Division of Chemistry and Biology, California Institute of Technology, Pasadena, California 91125; and Bell Laboratories, Murray Hill, New Jersey 07974

Contributed by John J. Hopfield, January 15, 1982

ABSTRACT **Computational properties of use to biological organisms or to the construction of computers can emerge as collective properties of systems having a large number of simple equivalent components (or neurons). The physical meaning of content-addressable memory is described by an appropriate phase space flow of the state of a system. A model of such a system is given, based on aspects of neurobiology but readily adapted to integrated circuits. The collective properties of this model produce a content-addressable memory which correctly yields an entire memory from any subpart of sufficient size. The algorithm for the time evolution of the state of the system is based on asynchronous parallel processing. Additional emergent collective properties include some capacity for generalization, familiarity recognition, categorization, error correction, and time sequence retention. The collective properties are only weakly sensitive to details of the modeling or the failure of individual devices.**

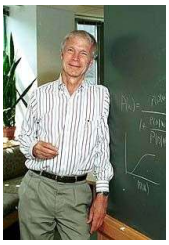
Given the dynamical electrochemical properties of neurons and

calized content-addressable memory or categorizer using extensive asynchronous parallel processing.

The general content-addressable memory of a physical system

Suppose that an item stored in memory is "H. A. Kramers & G. H. Wannier *Phys. Rev.* **60**, 252 (1941)." A general content-addressable memory would be capable of retrieving this entire memory item on the basis of sufficient partial information. The input "& Wannier, (1941)" might suffice. An ideal memory could deal with errors and retrieve this reference even from the input "Vannier, (1941)". In computers, only relatively simple forms of content-addressable memory have been made in hardware (10, 11). Sophisticated ideas like error correction in accessing information are usually introduced as software (10).

There are classes of physical systems whose spontaneous behavior can be used as a form of general (and error-correcting)

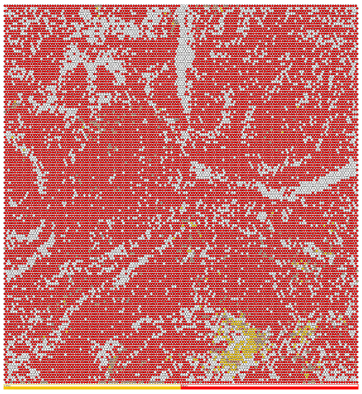


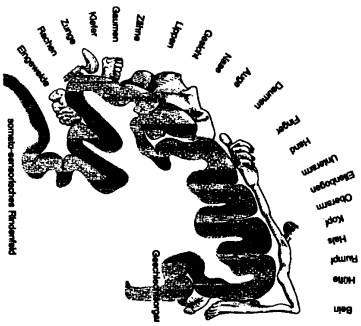
https://de.wikipedia.org/wiki/Ising-Modell

Neuronale Netze verhalten sich wie Spin-Gläser.
Spingläser ordnen sich spontan und immer.

Simulation eines Isingmodelles

Teuvo Kohonens selbstorganierende Karten





Viele Implementierungen in den 1980er und -90er Jahren

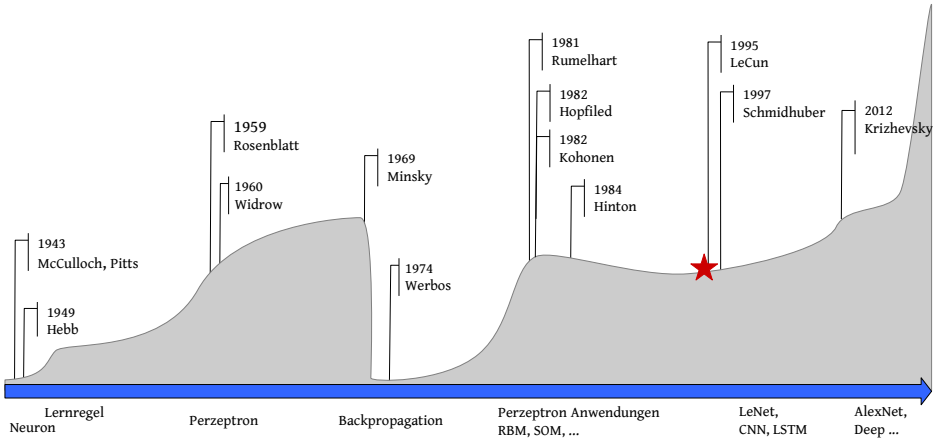
- Viele basieren auf:
- McCulloch-Pitts Neuronen
- Backpropagation Lernen

In den folgenden Jahrzehnten werden künstliche neuronale Netze eingesetzt für

- Aufgaben der schwachen KI:
- Klassifizierungen
- Mustererkennung

2.8 LeCun, Schmidhuber, Hinton: Deep Learning

Hypes and Stagnations



1995: Yann LeCun, LeNet

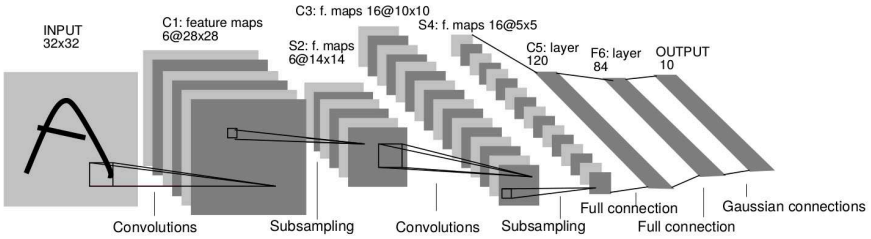
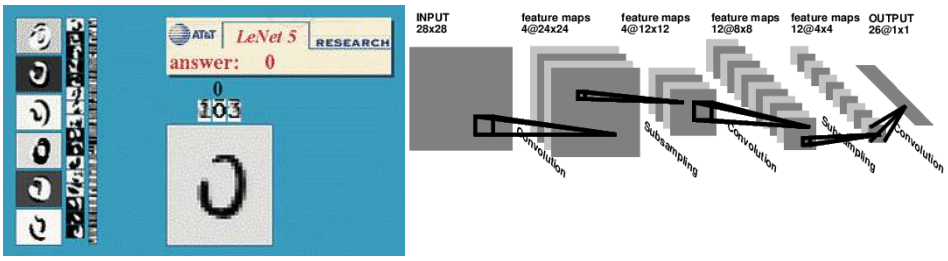


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner, 1998. Gradient-Based Learning Applied to Document Recognition.
Proc. IEEE 86, 2278–2324. http://yann.lecun.com/exdb/lenet/

Convolutional Neural Network (CNN)



Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik. *Comparison of learning algorithms for handwritten digit recognition.* In F. Fogelman and P. Gallinari, editors, *International Conference on Artificial Neural Networks* (1995) 53-60, Paris, EC2 & Cie.

2011: Schmidthuber, DanNet, erste GPU-Implementierung

Auf der IJCNN, 2011

J. Schmidhuber, *International Joint Conference on Neural Networks*, San Jose, California, (2011) USA, July 31 – August 5.

Convolutional Neural Network (CNN)

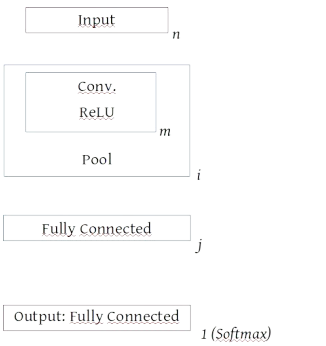
Ciregan, D., U. Meier, J. Schmidhuber. *Multi-Column Deep Neural Networks for Image Classification.* In *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012) 3642–49.

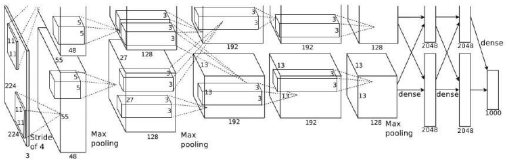
IJCNN 2011 ON-SITE TRAFFIC SIGN RECOGNITION COMPETITION OF 2 AUGUST 2011:
1ST (0.56% ERROR)
2ND HUMANS (1.16%)
3RD (1.69%)
4TH (3.86%)

Neuronales Netz

Andere ML- und Bildanalyseverfahren

2012: Krizhevsky, AlexNet, Durchbruch der GPU-Implementierung





Input	224x224x3 (Bilder mit RGB-Kanälen)
Conv.	96 Feature Maps, Kernel 11x11x3, Stride 4x4
Max pooling	5x5
Conv.	256 Feature Maps, Kernel 5x5x48
Max pooling	3x3
Conv.	384 Feature Maps, 3x3x256
Conv.	384 Feature Maps, 3x3x192
Conv.	256 Feature Maps, 3x3x192
Max pooling	3x3
FC	4096
FC	4096
Output	1000 (Klassen)

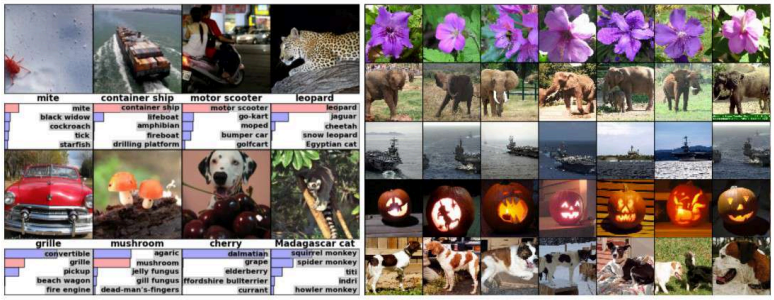
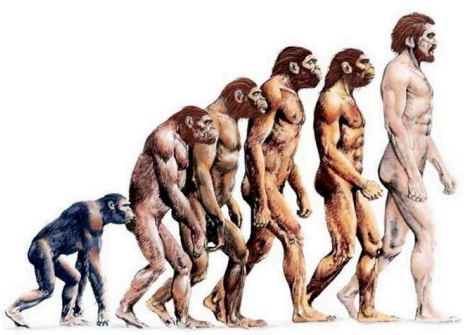


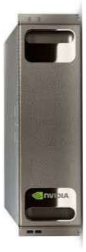
Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

2.9 Evolution der künstlichen Intelligenz



Catherine Pham, The theory of evolution

IBM PC

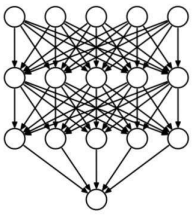


DGX

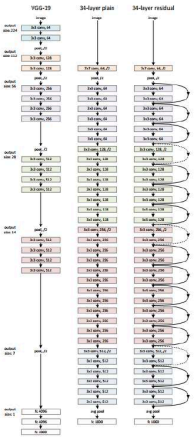
Quantencomputer

Technologie: Schnellere Computer!

1985:
4 Schichten,
16 Neurone,
55 Verbindungen



S. Nitish, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. J. Machine Learn. Res. **15** (2014) 1929–1958.



heute:
1000 Schichten,
500 000 Neurone,
100 000 000 Verbindungen

Sussillo, David, and L. F. Abbott. *Random Walk Initialization for Training Very Deep Feedforward Networks*. ArXiv:1412.6558 [Cs, Stat] (2014).

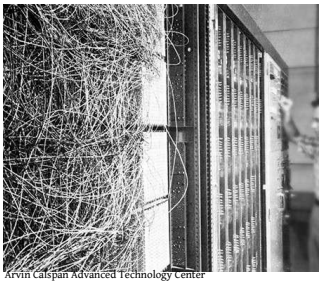
Technologie: Big Data!

14 197 122 Bilder
21 841 Synsets

Li Fei-Fei, image-net.org (2018) Stanford University

Technologie: Softwareengineering

1960
Hardware



Arvin Calspan Advanced Technology Center

1980
Software == Simulation

```
int  kr_makeDefaultUnit(void)
{
  struct Unit  *unit_ptr;
  FunctionPtr  func_ptr;
  int  unit_no;
  int  i;

  if ((unit_no = krm_getUnit()) == 0)
    return( KernelErrorCode );
  unit_no = abs(unit_no);

  if (KernelErrorCode != KRERR_NO_ERROR)
    return( KernelErrorCode );

  (void) kr_setAllUnitValues( unit_no, (FlintTypeParam) DEF_OUT, DefaultIAct,
                              DefaultIAct, DefaultBias );

  unit_ptr = unit_array + unit_no;

  unit_ptr->Ftype_entry = NULL;
  unit_ptr->value_a = (FlintType) 0;          /*previous bias change*/
  unit_ptr->value_b = (FlintType) 0;          /*previous bias slope*/
  unit_ptr->value_c = (FlintType) 0;          /*actual bias slope*/
```

Andreas Zell, SNNS, kernel.c

Leistung des NN = Leistung des Computers

MicroVax 3800,
DEC (1989)
1 CPU
12 MHz

Origin 300,
Silicon Graphics (1996)
32 CPUs
500 MHz

NVIDIA GeForce GTX 560,
nvidia (2011)

336 Kerne
1GHz

NVIDIA GeForce GTX 1070,
Pascal, nvidia (2016)

2423 Kerne
1,6 GHz

NVIDIA DGX-1, nvidia
(2016)

40 960 Kerne
5 120 Tensor-Recheneinheiten
3,6 GHz → 1000 TFLOP ($1000 * 10^{12}$)

1985: 100 Neurone

1995: 10 000 Neurone
(LeNet, Yann LeCun)

2012: 622 312
Neurone (AlexNet,
Alexander Krizhevsky)

2016
= 100 x

2018
= 10 000 x

```
function train(w, dtrn; lr=.05)
    for (x,y) in dtrn
        g = lossgradient(w, x, y)
        update!(w,g;lr=lr)
    end
    return w
end

function main()
    w = weights(o[:hidden]...; atype=atype, winit=o[:winit])
    xtrn,ytrn,xtst,ytst = mnist()
    global dtrn = minibatch(xtrn, ytrn, o[:batchsize]; xtype=atype)
    global dtst = minibatch(xtst, ytst, o[:batchsize]; xtype=atype)
    @time for epoch=1:o[:epochs]
        train(w, dtrn; lr=o[:lr], epochs=1)
        report(epoch)
        if o[:gcheck] > 0
            gradcheck(loss, w, first(dtrn)...; gcheck=o[:gcheck])
        end
    end
    return w
end
```

Knet → für Informatiker

```
(x_train, y_train), (x_test, y_test) = mnist.load_data()

model = Sequential()
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dense(512, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test)
```

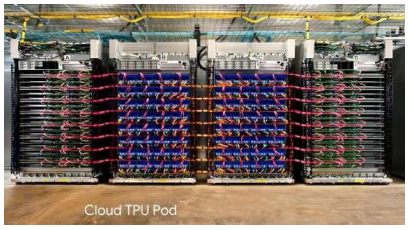
Keras → für jedermann

1985: 100 Neurone

1995: 10 000 Neurone (LeNet, Yann LeCun)

2012: 622 312 Neurone (AlexNet, Alexander Krizhevsky)

2018: AlphaZero, viele Neurone
64 TPUs 2. Generation
5000 TPUs 1. Generation
(7000 PFLOP)



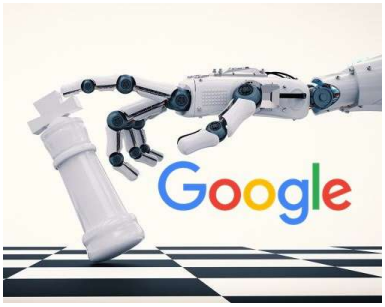
Google TensorFlow Precessing Unit, 3. Generation
Google (2018)

→ 100 PFLOP (100 * 10^{15})

= 1 000 000 x 

2016: AlphaGo

2018: AlphaZero



S. David, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, et al. *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm.* ArXiv:1712.01815v1 [Cs, AI] (2017).

Game	White	Black	Win	Draw	Loss
Chess	*AlphaZero*	*Stockfish*	25	25	0
	Stockfish	*AlphaZero*	3	47	0
Shogi	*AlphaZero*	*Elmo*	43	2	5
	Elmo	*AlphaZero*	47	0	3
Go	*AlphaZero*	*AG0 3-day*	31	–	19
	AG0 3-day	*AlphaZero*	29	–	21

Funktioniert wie 1985, aber

Schach: 44 000 000 Partien in 4 h

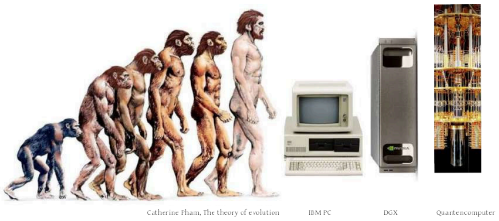
Großmeister: 50 Jahre, 2500 Partien/Tag
oder 12000 Jahre bei 10/Tag

Evolutionäre Sprünge in der Vergangenheit und in der Zukunft?

Evolution geschieht **nicht** linear

Sprünge bei:

- Draht ⟶ Simulation
- Computer ⟶ GPU
- GPU ⟶ spezielle ML-Hardware
- Programmieren ⟶ Konfiguration



Catherine Pham, The theory of evolution IBM PC DGX Quantencomputer

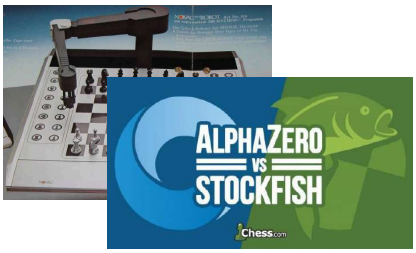
Starke KI

86 000 000 000 Neurone

Wann?

100 x 100

Schwache KI



... wird stärker ...

10 000 000 Neurone

Pixabay

- 1970er: 50 – 200 Interessierte
- 1980er/90er: steigendes Interesse
- Heute: jeder!

Fragen:

- Wann ist die schwache KI so stark wie die starke?
- Wann ist sie stärker?
- Wie verhindern wir Mißbrauch, wenn es jeder kann?

  Ban lethal autonomous weapons:
  https://autonomousweapons.org/slaughterbots/

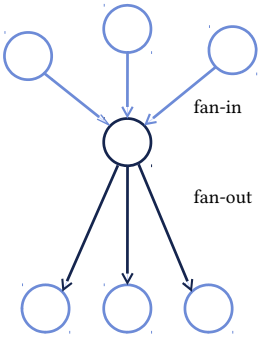
  US Air Force, 2017:
  https://www.youtube.com/watch?v=PYLP0pAGbE0#t=0m40s

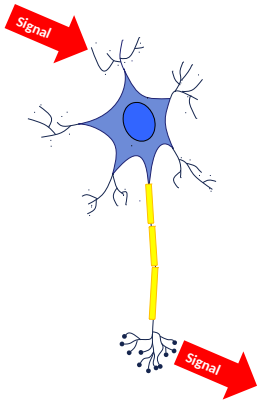
Kapitel 3:

Das Perzeptron

3.1 Biologische und künstliche Neurone

Der Aufbau unserer künstlichen Neurone ist etwas vereinfacht ...



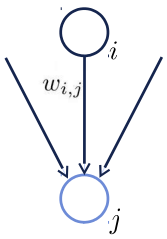


Die Aktivierung biologischer Neurone wird mit Hilfe der `net()`–Funktion und der Aktivierungsfunktion simuliert:

$$net_j = f\left(o_{i..n}, w_{i..n,j}\right) = \sum_i o_i \cdot w_{ij}$$

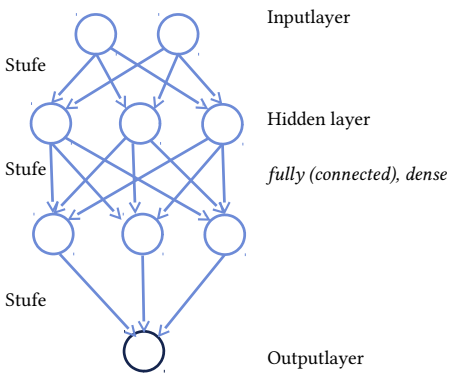
$$\theta_j$$

$$a_j(t+1) = f_{act}\left(a_j(t), net_j(t), \theta_j\right)$$



3.2 Nomenklatur

Namen im DNN:



3.3 Plattformen und Sprachen

1. Generation:

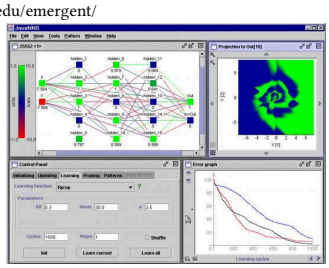
Simulatoren mit GUI

- SNNS, JavaNNS
- http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/
- RSNNS

Emergent

- https://grey.colorado.edu/emergent/

memBrain



Viele, viele Alternativen!

Tensorflow (Google)
Go, Python, Julia, ...

OpenNN (NeuralDesigner)
C++ Library

Keras (TensorFlow)
Python Library

Mxnet
Deep learning library: Python, R, Scala, Julia

DeepLearning4J
Java – rasante Entwicklung!

KNet
Julia, Idee statt Framework

Torch7
Tensor-Library (MATLAB-Stil/Lua)

PyTorch
Tensor-Library (AutoGrad)

Caffe
C++, PyCaffy/MATCaffe

Lasagne (Backend Theano)
Deep learning, Python

Mocha
Deep learning (Caffe) für Julia

MatConvNet
Matlab

Manche Sprachen sind schneller als andere - interessanterweise werden sehr langsame Sprachen auch im Machinine Learning genutzt:

https://juliakorea.github.io/benchmarks/

	C gcc 4.8.5	Julia 0.6.0	LuaJIT scilua v1.0.0-b12	Fortran gcc 4.8.5	Go go1.7.5	Java 1.8.0_14
iteration_pi_sum	1.00	1.00	1.00	1.00	1.02	1.01
recursion_fibonacci	1.00	1.88	1.40	0.58	1.88	1.71
recursion_quicksort	1.00	0.94	1.53	1.30	1.24	2.55
parse_integers	1.00	1.35	1.00	5.39	1.00	2.49
print_to_file	1.00	0.66	0.57	3.44	1.98	6.01
matrix_statistics	1.00	1.74	1.63	1.87	5.77	4.91
matrix_multiply	1.00	0.98	1.11	1.27	1.24	8.85
userfunc_mandelbrot	1.00	0.76	1.04	0.75	0.81	1.13

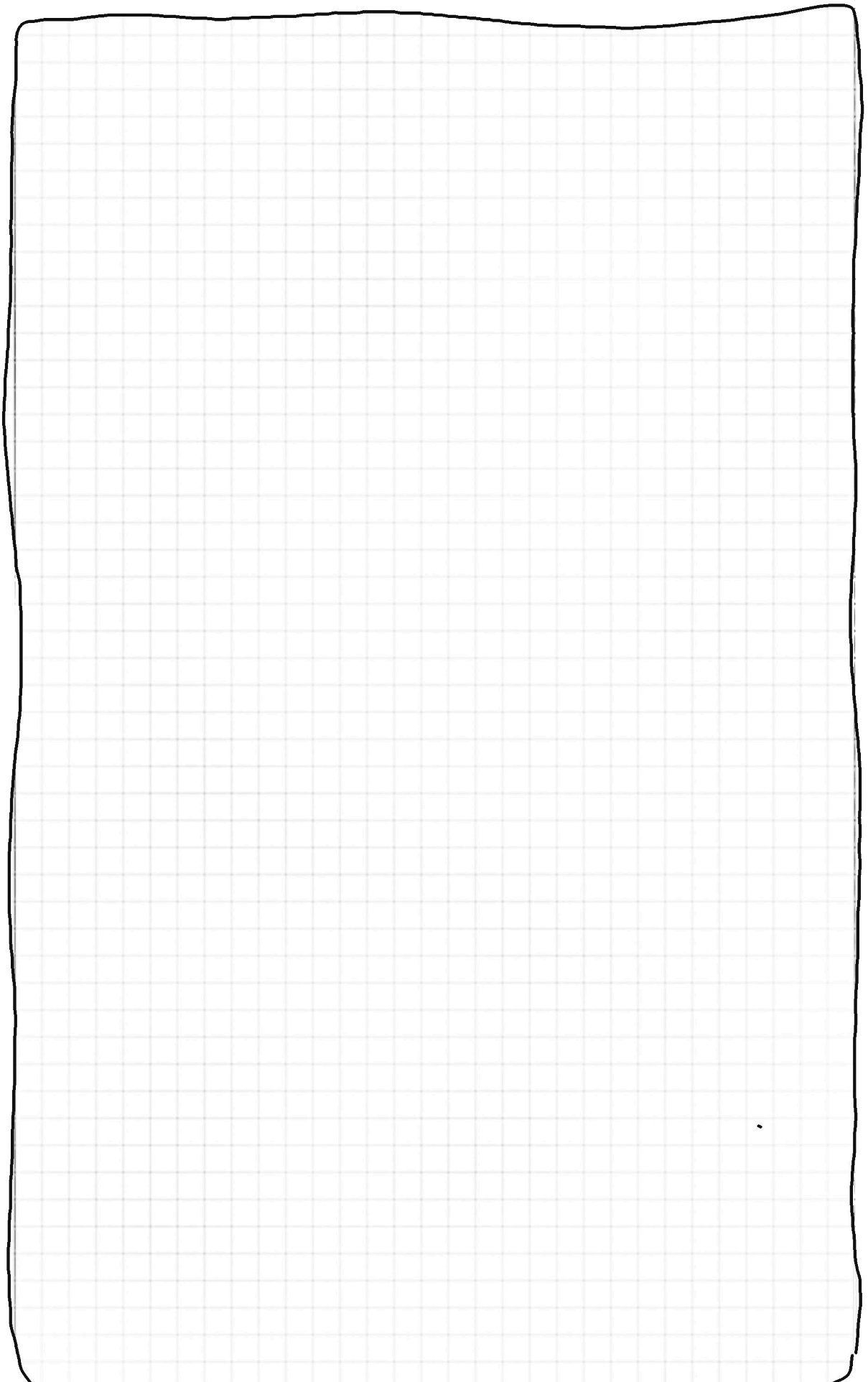
	JavaScript V8 4.5.103.47	Matlab R '2017a'	Mathematica 11.1.1	Python 3.5.4	R 3.3.1	Octave 4.0.3
iteration_pi_sum	1.00	1.00	1.54	20.25	8.88	448.22
recursion_fibonacci	3.82	20.29	142.28	98.69	652.86	17878.75
recursion_quicksort	2.90	3.08	45.53	37.51	263.32	3261.04
parse_integers	6.41	238.10	14.72	18.77	50.90	3797.82
print_to_file	--	116.70	61.93	1.38	136.95	197.32
matrix_statistics	3.06	17.86	7.55	17.78	19.65	56.20
matrix_multiply	24.89	1.18	1.19	1.17	8.84	1.21
userfunc_mandelbrot	1.07	19.48	19.35	142.95	347.41	8981.32

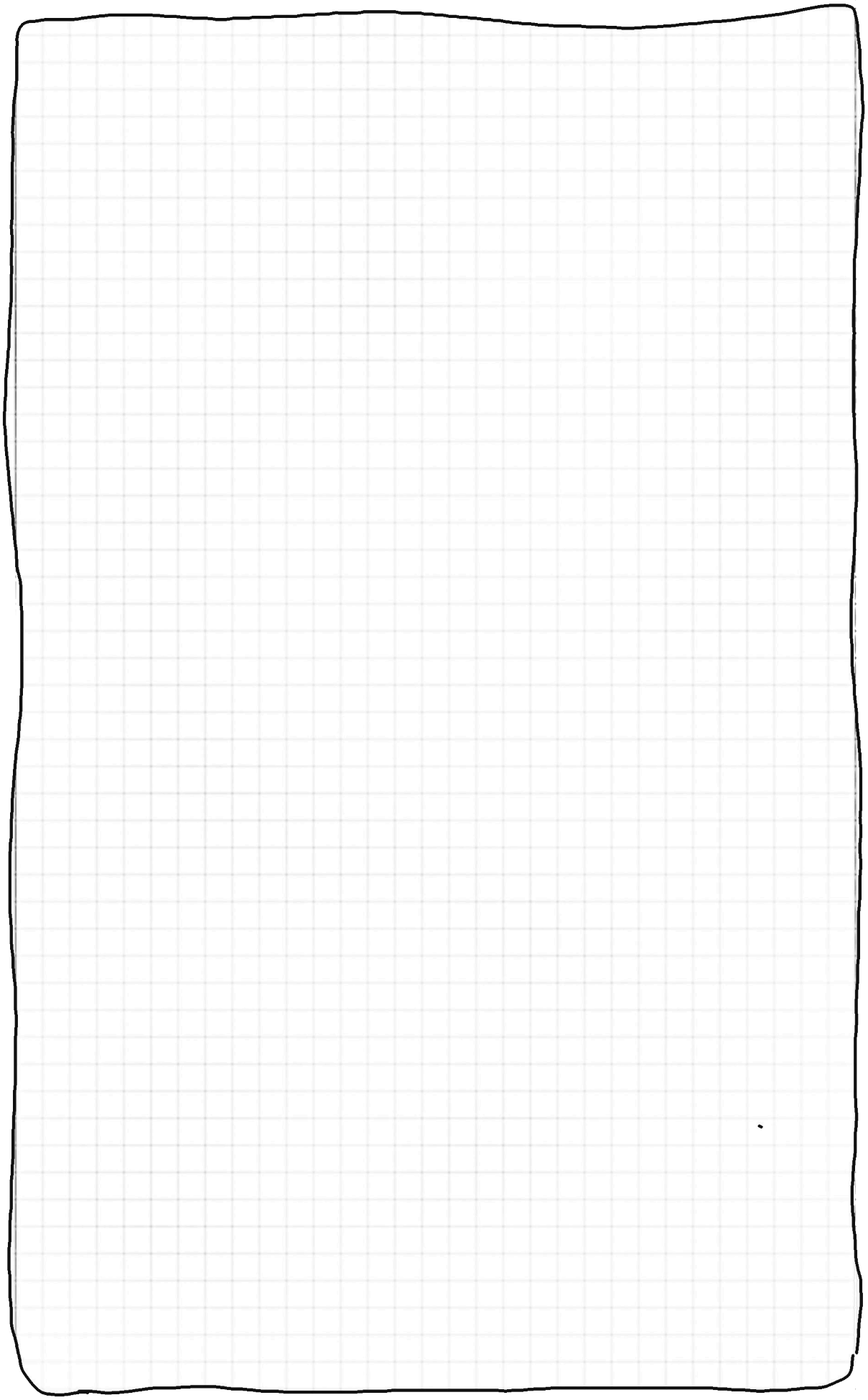
Mini-Cheat-Sheet für Umsteiger:

Numpy	Matlab	R	Julia	
`x[0]`	`x(1)`	`x[1]`	`x[1]`	Erstes Element
`np.random.randn(3,3)`	`randn(3)`	`Matrix( rnorm(9),3)`	`randn(3,3)`	3x3 Random-Matrix mit normalverteilt
`np.arange(1,11)`	`1:10`	`1:10`	`1:10`	1,2,...,10
`X * Y`	`X .* Y`	`X * Y`	`X .* Y`	Elementweise Multiplikation
`np.dot(X,Y)`	`X * Y`	`X %*% Y`	`X * Y`	Matrixmultiplikation
`d = {'one':1,'two':2}` `d['one']`	`d = containers.Map( {'one','two'}, {1,2});` `d('one')`	-	`d = Dict( "one"=>1, "two"=>2)` `d["one"]`	Hashtable
`f = lambda x, mu, sigma: np.exp(-(x-mu)**2 / (2*sigma**2) ) / sqrt(2*np.pi*sigma^2)`	`f = @(x,mu,sigma) exp(-(x-mu)^2 (2*sigma^2)) / sqrt(2*pi*sigma^2)`	`f = dnorm( x, mu, sd)`	`f(x,μ,δ) = exp( -(x-μ)^2/2δ^2)/ sqrt(2π*δ^2)`	Gaussian density function
`r = np.random.rand( *x.shape)` `y = x * (r > t)`	`r = rand(size(x))` `y = x .* (r > t)`	`y = ifelse( runif(length(x)) > t, x, 0)`	`r = rand(size(x))` `y = x .* (r .> t)`	Dropouts

Man kann DNNs programmieren, dann ist man ein Ingenieur - oder verwenden, dann ist man ein Autofahrer. Aber: wer ein besonders guter Autofahrer sein will, der muss wissen wie sein Auto funktioniert!

3.4 Neuronale Netz sind konvexe Klassifizierer





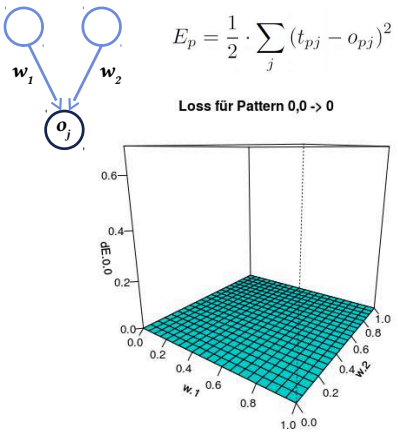
3.5 Supervised Training neuronaler Netze

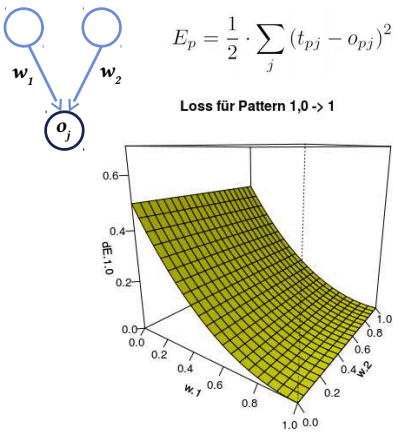
Das Perzeptron lernt überwacht:

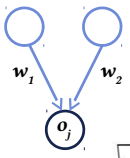
- Das korrekte Ergebnis wird vorgegeben und mit der aktuellen Ausgabe verglichen.
- ***Ein Fehler wird berechnet.***
- ***Parameter werden angepasst.***
- Das neuronale Netz wird so lange trainiert, bis die Outputneurone die gewünschte Ausgabe erzeugen.

Entspricht das dem biologischen Vorbild?

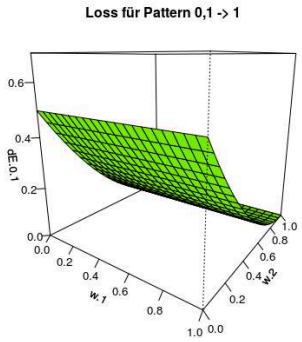
Fehlerfläche am Beispiel OR für die 4 Pattern:

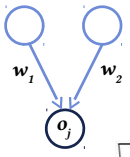




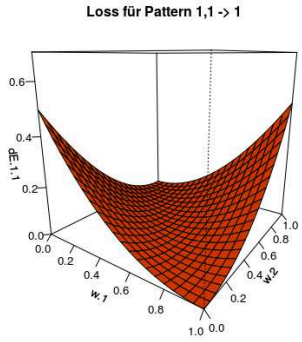


$$E_p = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2$$

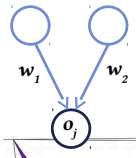




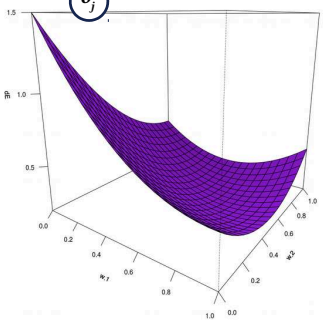
$$E_p = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2$$



... und für alle 4 zusammen



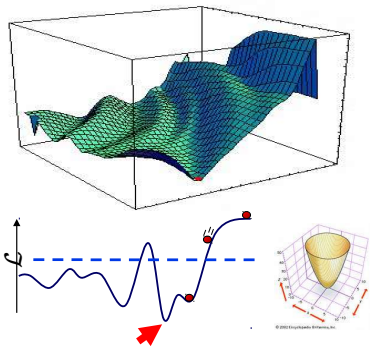
$$E_p = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2$$



Loss für alle Patterns:

$$\mathcal{L}^{Square} = E = \sum_p E_p$$

Aus der Fehlerfläche können wir den Gradienten ermitteln:

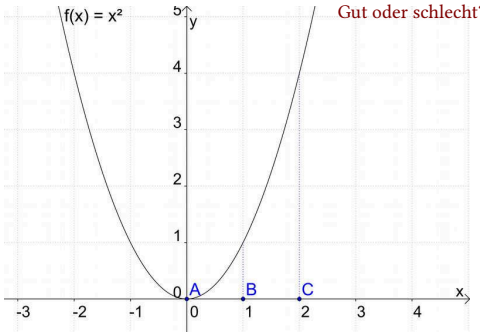


Lokal: immer eine Parabel! $E_p = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2$

text

großer Gradient ⟶ große Schrittweite
kleiner Gradient ⟶ kleine Schrittweite

Gut oder schlecht?



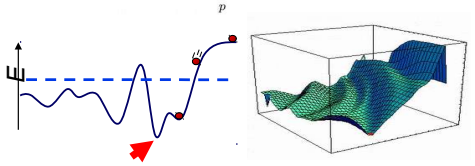
3.6 Ableitung der Deltaregel für ein einstufiges Netz

Fehler des Netzes:

- Differenz zwischen tatsächlichem Output (oj) und Teaching Input (tj) jedes Outputneurons
- Abhängig von allen Gewichten (wij)
- Ergibt eine Hyperfläche mit i x j Dimensionen

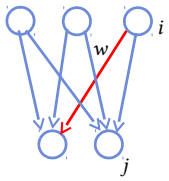
$$E_p = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2$$

$$\mathcal{L}^{Square} = E = \sum_p E_p$$



mehr Nomenklatur, die in der Veranstaltung konsistent gehalten wird!

$$\Delta w_{ij} = -\eta \frac{\partial}{\partial w_{ij}} E(W) = -\eta \cdot \sum_p \frac{\partial}{\partial w_{ij}} E_p$$



Indices:

i: Quelle

j: Ziel

p: Pattern/ Sample

Symbole:

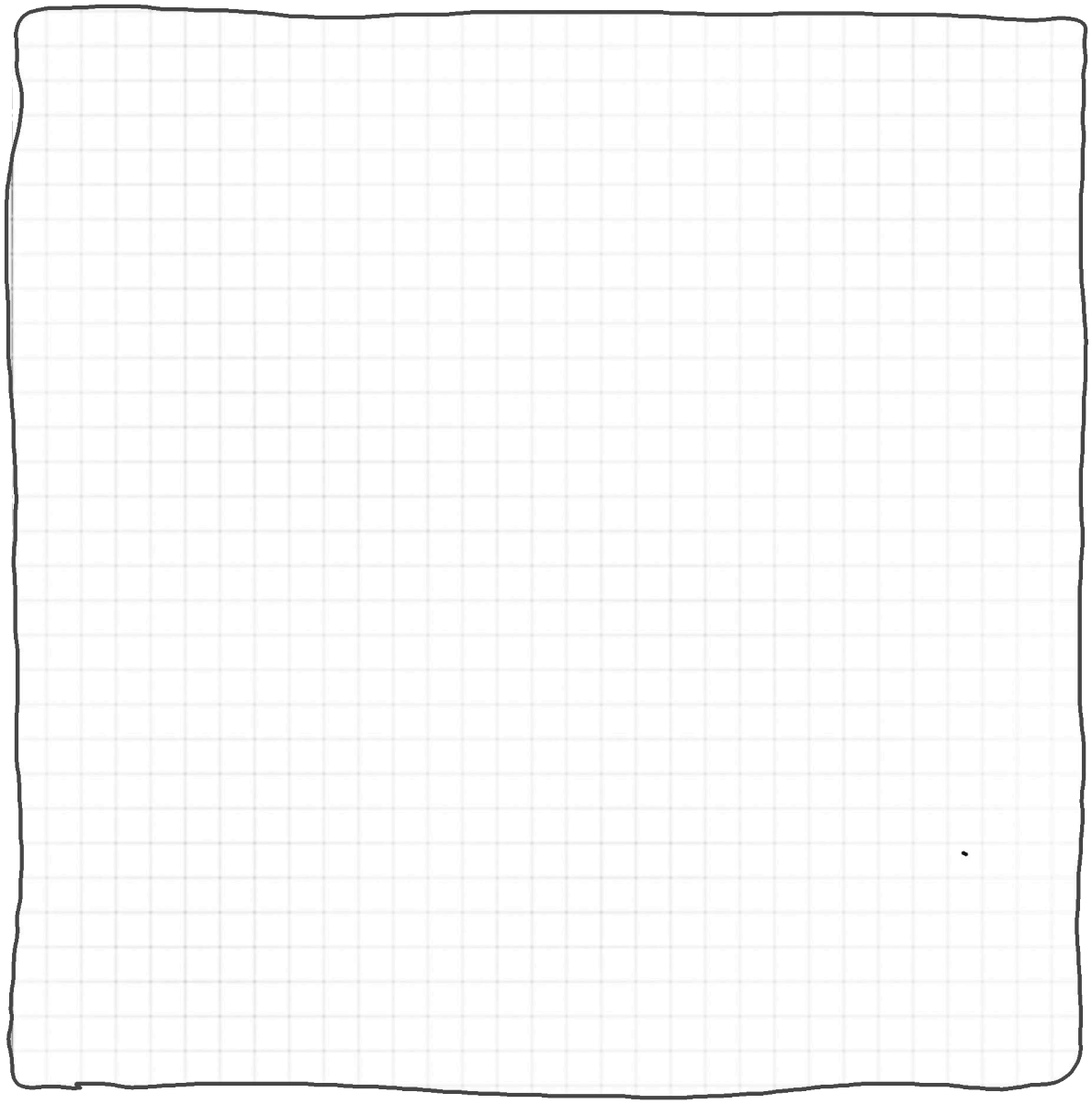
o: Output/ Aktivierung

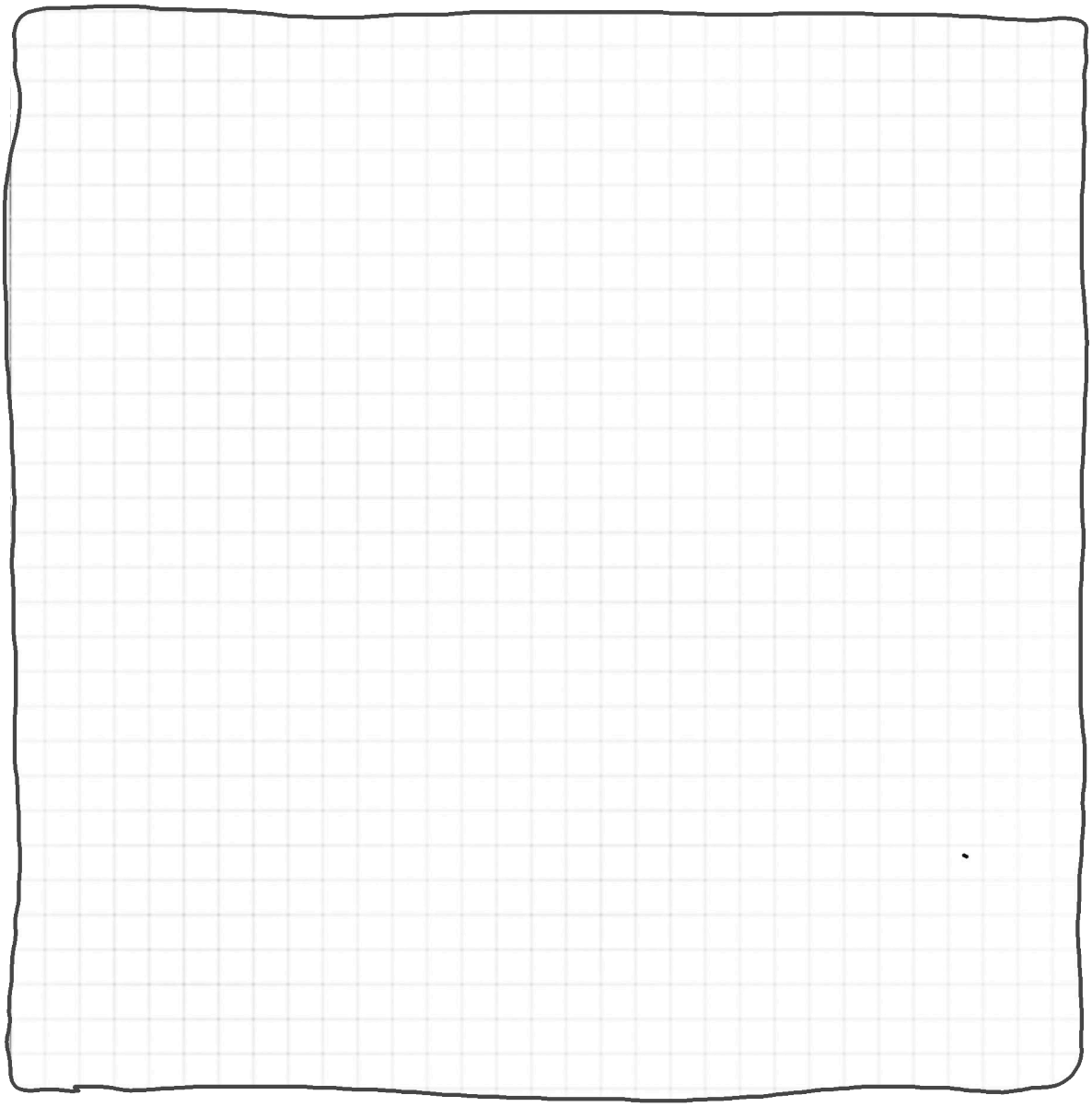
t: Teaching input

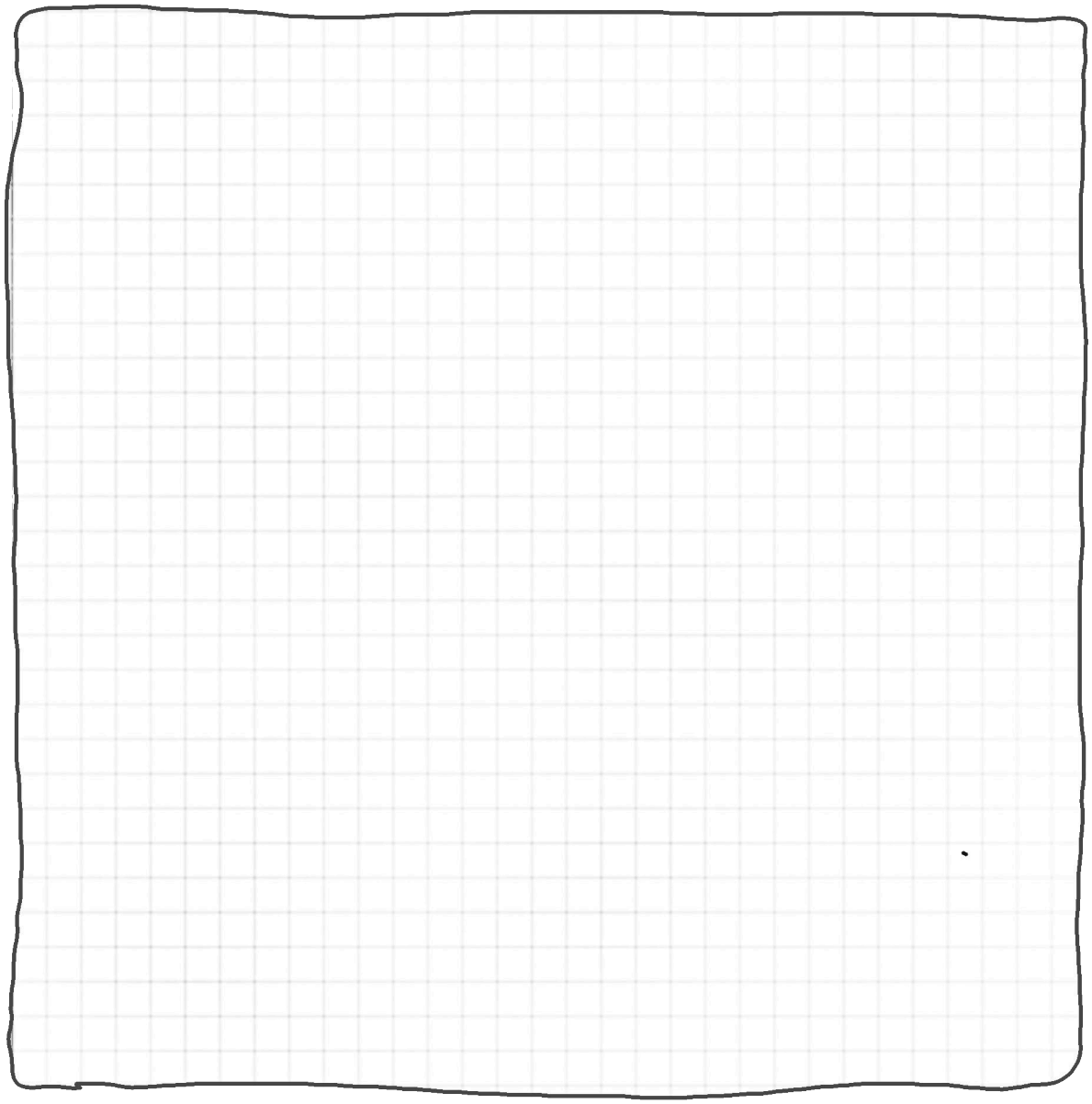
η: Lernarate eta

$$\Delta w_{ij} = -\eta \cdot \sum_p \frac{\partial E_p}{\partial w_{ij}}$$

$$E_p = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2$$

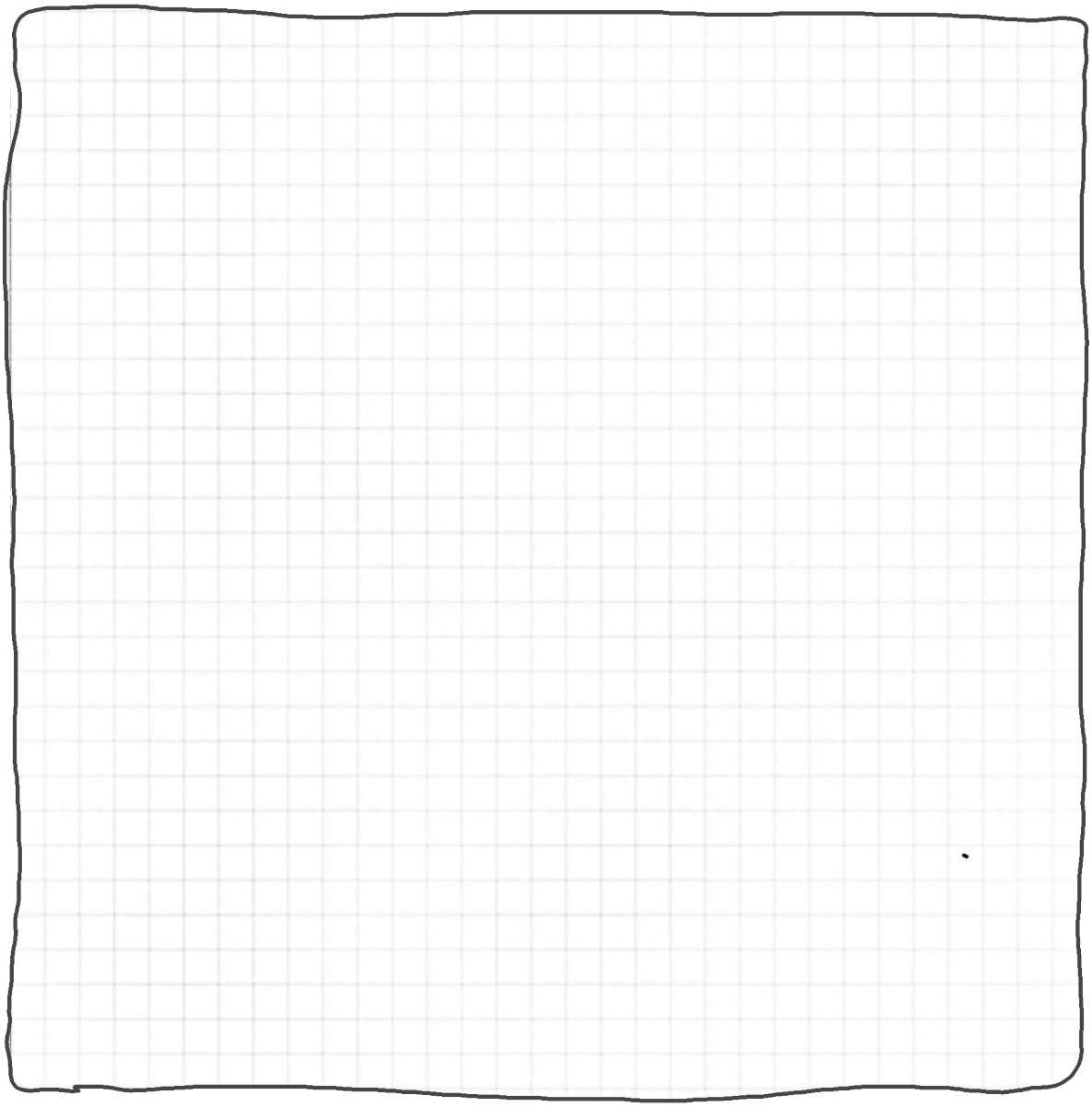






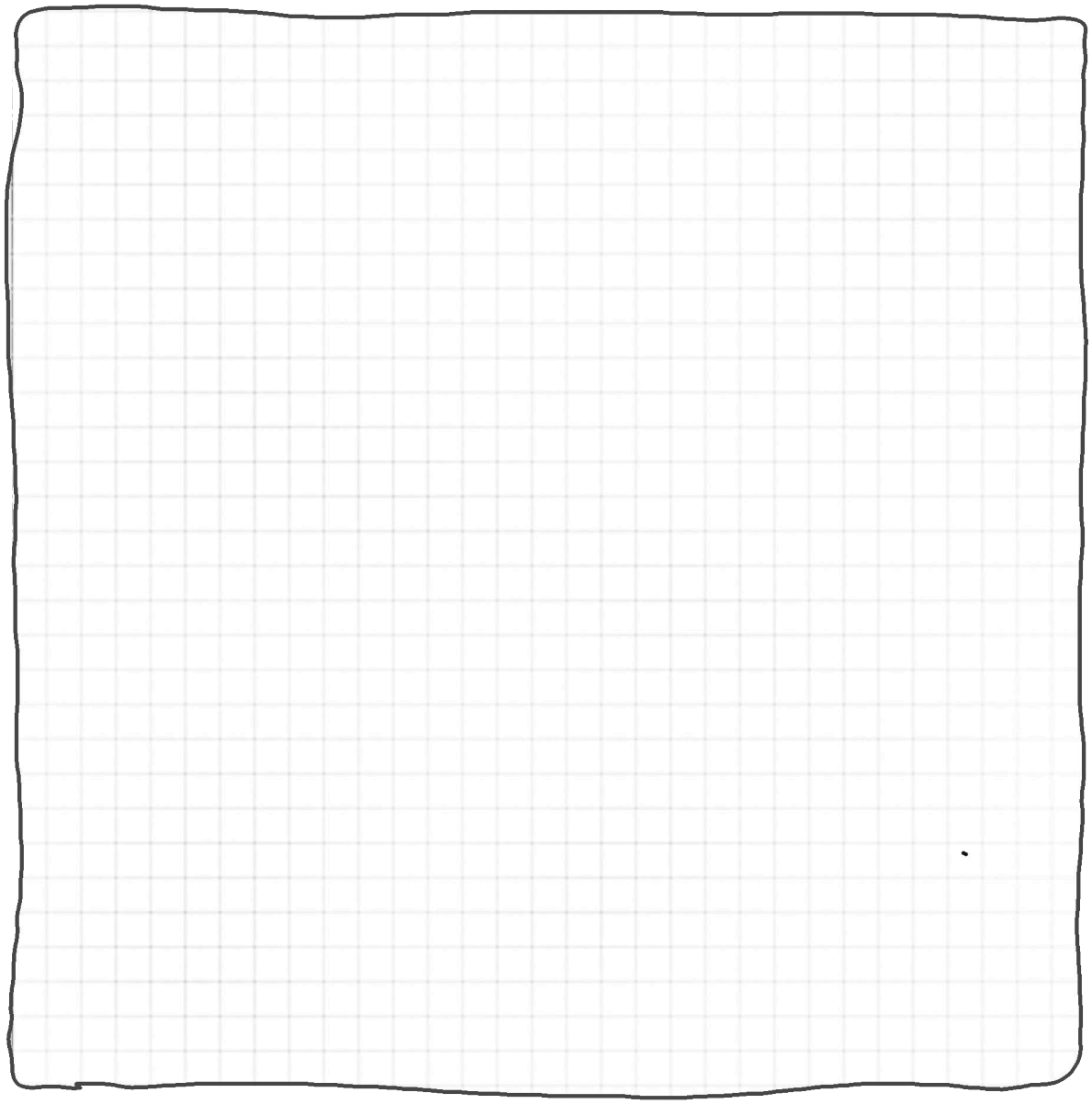
Deltaregel/Widrow-Hoff-Regel als offline-Variante

$$\Delta w_{ij} = \eta \cdot \sum_p o_{pi} \delta_{pj} = \eta \cdot \sum_p o_{pi} \left(t_{pj} - o_{pj} \right)$$

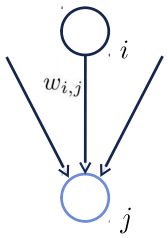


Deltaregel/Widrow-Hoff-Regel als online-Variante

$$\Delta w_{pij} = \eta o_{pi} \delta_{pj} = \eta o_{pi} \left(t_{pj} - o_{pj} \right)$$



ABER: die Aktivierungsfunktion dürfen wir nicht vergessen



bisher: $\frac{\partial E(o(w))}{\partial w} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial w}$

aber:

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})$$

und

$$o_{pj} = f_{act}(\sum_i (o_i \cdot w_{ij}))$$

deshalb:

$$\frac{\partial E(o(net(w)))}{\partial w} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w}$$

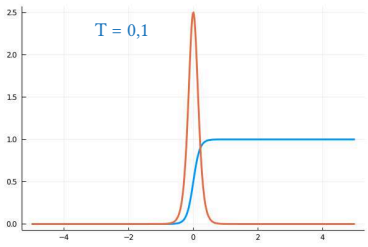
$$\Delta w_{pij} = \eta o_{pi} \delta_{pj} \quad \frac{d f_{act}}{d(net)}$$

$$\frac{df(x)}{d(x)} = r \cdot f(x)\left(1 - f(x)\right)$$

Logistische Gleichung

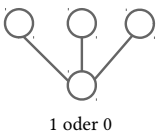
$$\Delta w_{pij} = \eta \cdot o_{pi} \cdot \frac{d f_{act}}{d(net)} \cdot \delta_{pj}$$

$$\Delta w_{pij} = \eta \cdot o_{pi} \cdot o_{pj}(1 - o_{pj}) \cdot \delta_{pj}$$



3.7 Kreuzentropie als Loss

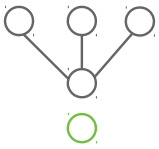
Binäre Kreuzentropie



Shannon Entropie
(Informationsgehalt bei *zwei* möglichen Mustern P_1 und P_2):

$$H = -\sum_p (P_p \cdot log_2 P_p)$$

$$H = -(P_1 \cdot log_2 P_1 + P_2 \cdot log_2 P_2)$$

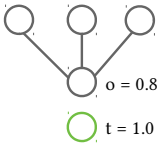


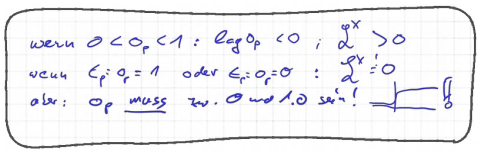
Kreuzentropie:
(Informationsgehalt *des Unterschieds* bei zwei Klassen):

$$\mathcal{L}^{Xbinary} =$$

$$-\sum_p (t_p \cdot \log_2(o_p) + (1 - t_p) \cdot \log_2(1 - o_p))$$

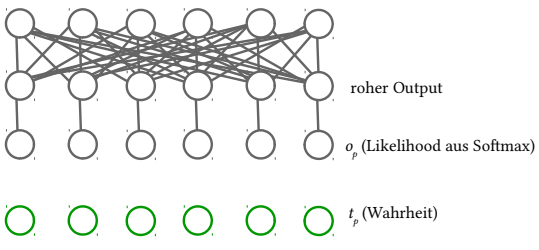
Die Kreuzentropie hat alle Eigenschaften, die wir von einer Lossfunktion erwarten!





$$\mathcal{L}^{Xbinary} = -\sum_p (t_p \cdot \log_2(o_p) + (1 - t_p) \cdot \log_2(1 - o_p))$$

Bei mehreren Outputneuronen geht das ganz ähnlich:

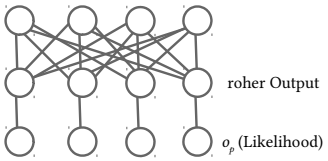


$$\mathcal{L}_p^X(o,t) = \sum_j t_{pj} \cdot \log_2 \frac{1}{o_{pj}} = -\sum_j t_{pj} \cdot \log_2 o_{pj}$$

$$\mathcal{L}_p^X(o,t) = -(t_{p,1} \cdot \log_2 o_{p,1} + t_{p,2} \cdot \log_2 o_{p,2} + \cdots + t_{p,j} \cdot \log_2 o_{p,j})$$

Und wenn man denkt eine One-Hot-Kodierung zu brauchen, dann braucht man sie (fast) sicher nicht!

$$\mathcal{L}_p^X(o,t) = -(t_{p,1} \cdot \log_2 o_{p,1} + t_{p,2} \cdot \log_2 o_{p,2} + t_{p,3} \cdot \log_2 o_{p,3} + t_{p,4} \cdot \log_2 o_{p,4})$$



o = (0.3, 0.6, 0.1, 0.0)
t = (0.0, 1.0, 0.0, 0.0)

$$\mathcal{L}_p^X(o,t) = -t_{p,2} \cdot \log_2 o_{p,2} = -\log_2 o_{p,j=class}$$

```
julia> t = 2
julia> x_loss(o,t) = -log(o[t])
# o: Vektor der Outputs
# t: Index der korrekten
#    Klasse
```

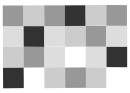
Die Berechnung der Kreuzentropie als NLL (nagative-Log-Likelihood) für die ganze Minibatch ist einfach:

... aus unnormalisiertem Output (== ***keine*** Aktivierungsfunktion!):

kein One Hot!

```
# o: Vektor der Outputs
# t: Teaching-Vektor mit korrekten Klassen
# für Minibatch der Größe n:
julia> p = exp.(o) ./ sum(exp.(o), dims=1)
julia> nll = -1/n * sum([log(p[t[i],i]) for i in 1:n])
```

o, p:

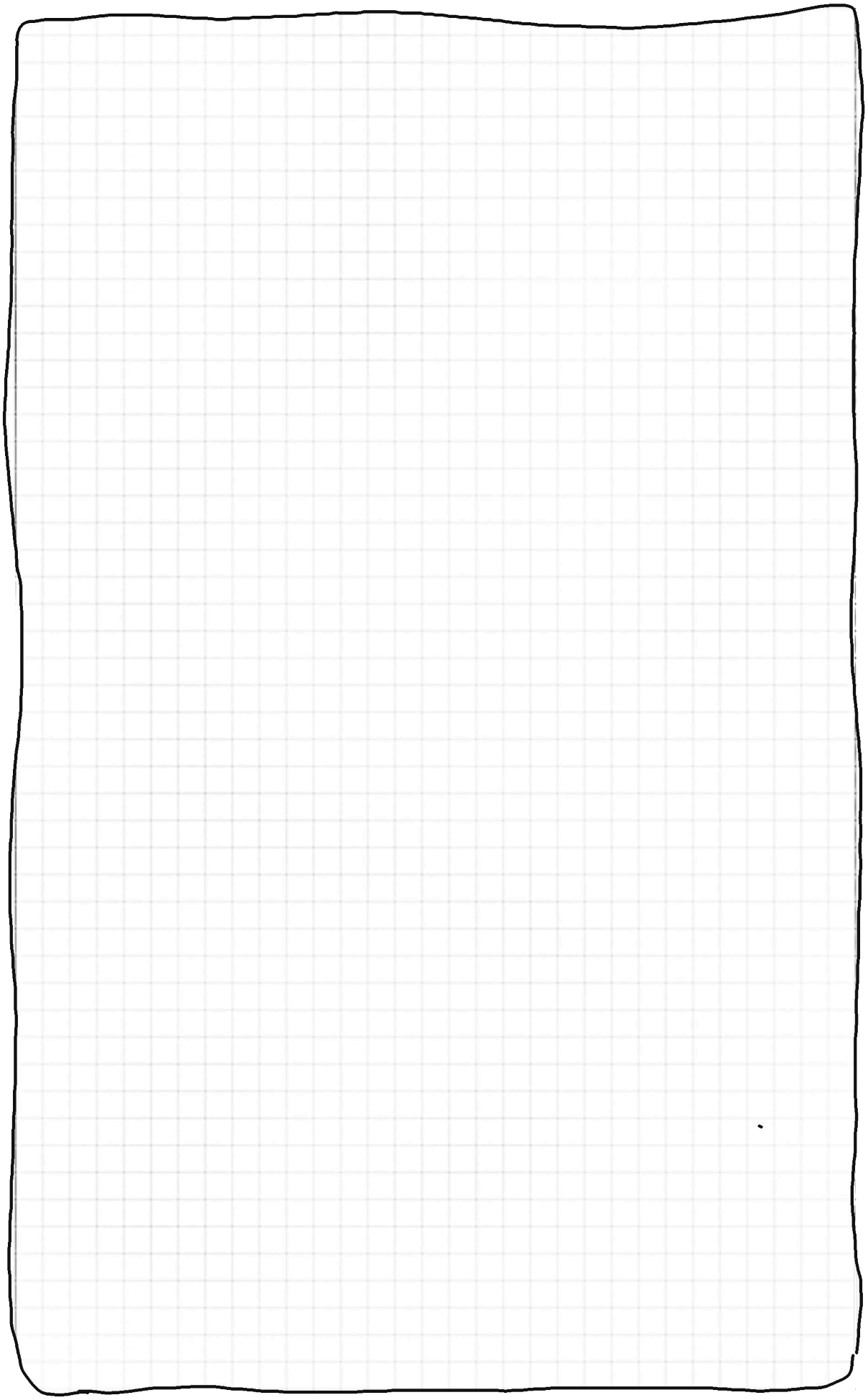


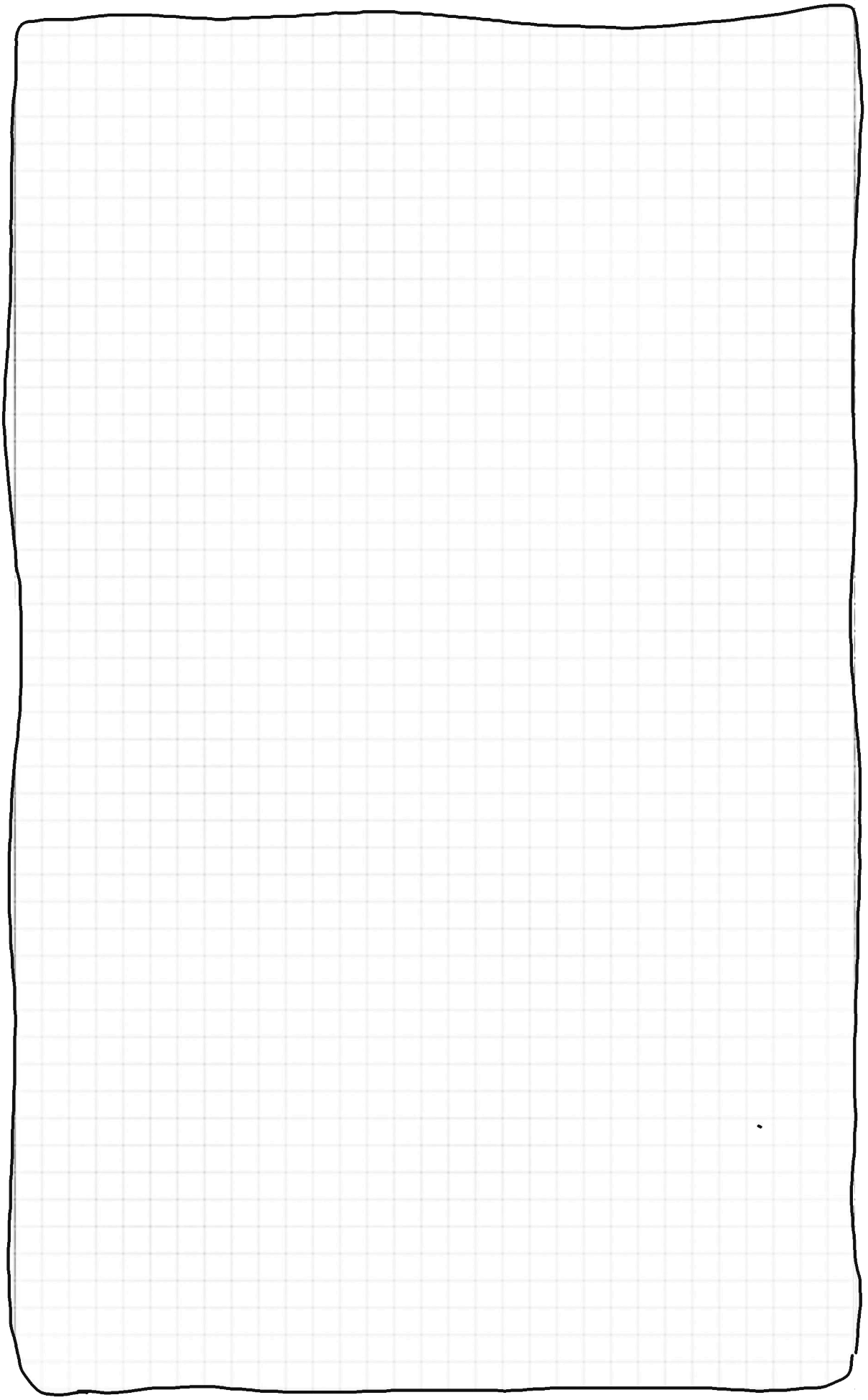
t: [1, 3, 1, 4, 2, 1]

3.8 Ableitung der Deltaregel für ein einstufiges Netz mit Kreuzentropie als Loss

Lernregel für die binäre Kreuzentropie

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} - \sum_p (t_p \cdot \log(o_p) + (1 - t_p) \cdot \log(1 - o_p))$$





Zusammenfassung: Lernregel für Square-Loss

Aktivierungsfunktion:

- linear:

$$\Delta w_{pij} = \eta \cdot o_{pi} \cdot \delta_{pj}$$

- sigmoid:

$$\Delta w_{pij} = \eta \cdot o_{pi} \cdot o_{pi}(1 - o_{pi}) \cdot \delta_{pj}$$

Zusammenfassung: Lernregel für X-Entropy-Loss

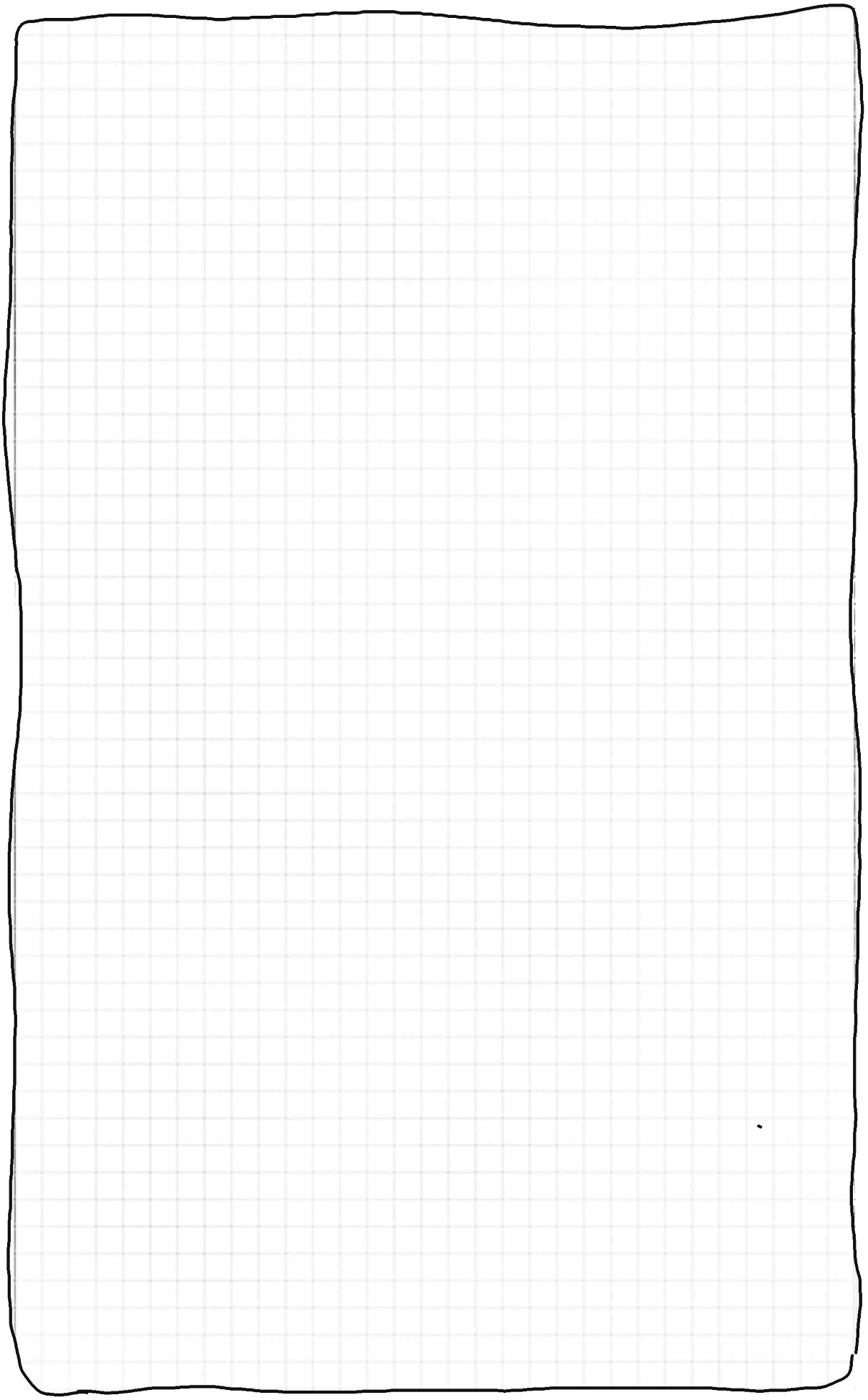
Aktivierungsfunktion:

- linear:

$$\Delta w_{pij} = \eta \cdot o_{pi} \cdot \frac{\delta_{pj}}{o_{pi}(1 - o_{pi})}$$

- sigmoid:

$$\Delta w_{pij} = \eta \cdot o_{pi} \cdot \delta_{pj}$$



Kapitel 4:

Das Multilayer-Perzeptron (MLP)

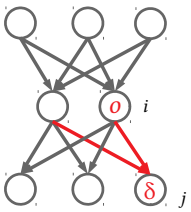
4.1 Backpropagation

Schritt 1: Forwardpropagation:

$$\Delta w_{pij} = -\eta \cdot o_{pi} \cdot \delta_{pj}$$

$$\delta_{pj} = t_{pj} - o_{pj}$$

Training der letzten Schicht:

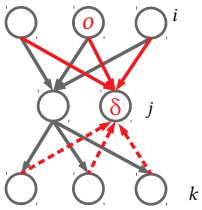


Schritt 2: Backpropagation:

$$\Delta w_{pij} = -\eta \cdot o_{pi} \cdot \delta_{pj}$$

$$\delta_{pj} = \sum_k \delta_{pk} w_{pk}$$

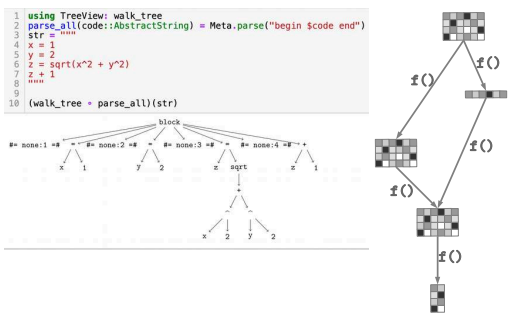
Training der **vor**letzten Schicht:

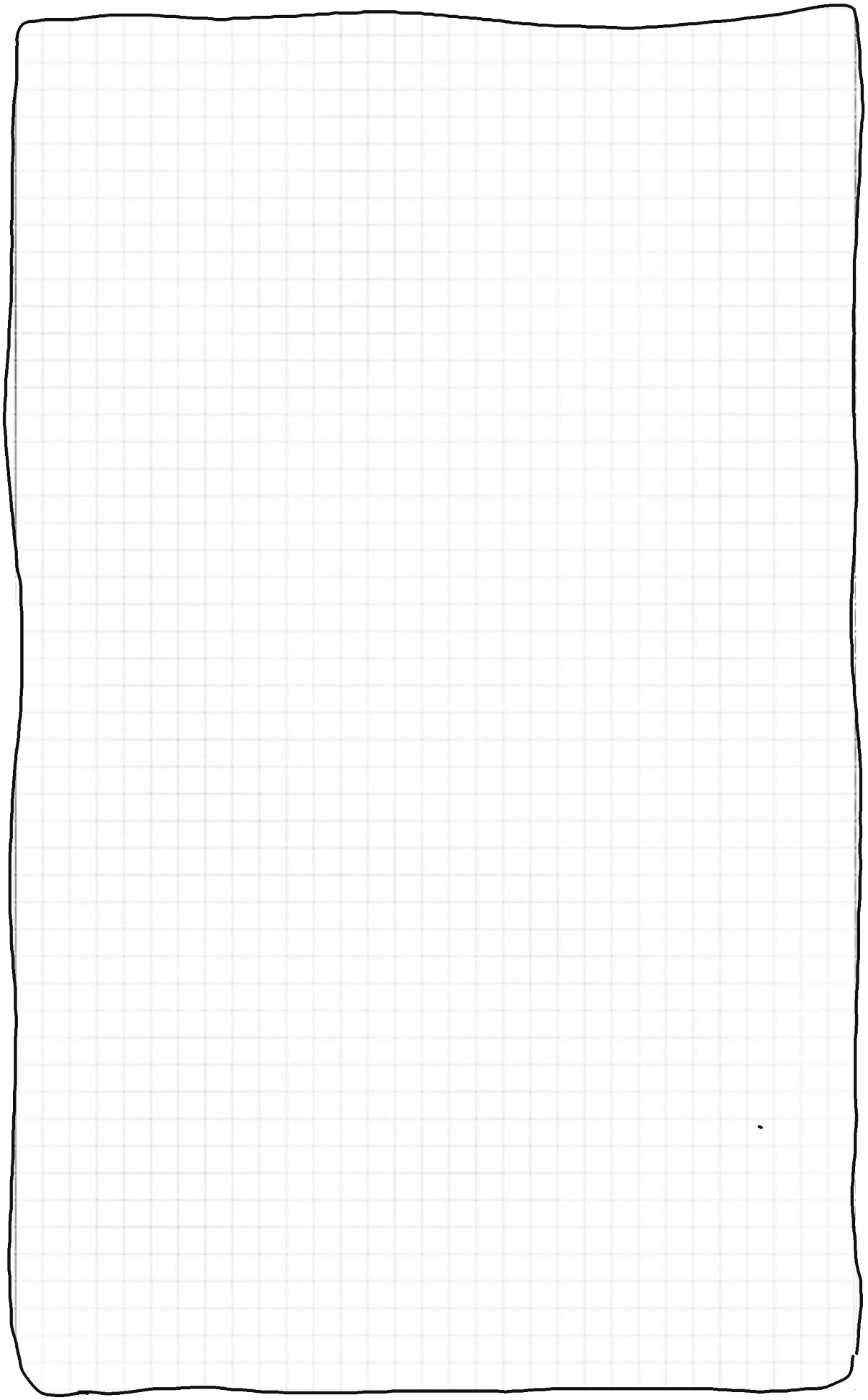


4.2 Automatisches Differenzieren

AutoGrad führt den Code *unter Beobachtung* aus und schriebt ein Protokoll (ein *Tape*) mit. Das Tape kann mit den bekannten Ableitungsregel abgeleitet werden.
AutoGrad:

AST und Berechnungsgraph (Tape):





Kapitel 5:

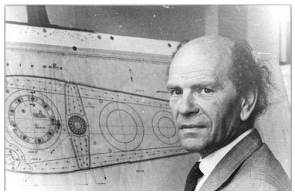
Regularisierung

5.1 Lernen und Verstehen

Es gibt Streber und Ingenieure, was ist unser Netz?

- Wie lernt ein Streber?

- Wie lernt ein intelligenter Mensch?



5.2 Generalisierung und Overfitting

Generaliserung

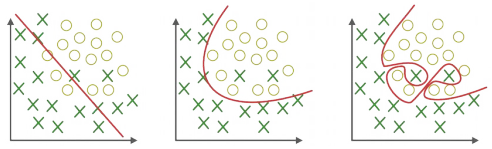
Neuronale Netze sollen intelligent sein – keine Streber!

Unser NN soll ...

- ... das Problem zu repräsentieren.
- ... nicht die Trainingsmuster auswendig lernen.
- ... mit unbekannten Daten umgehen können.
- ... im Notfall halbwegs gut raten!

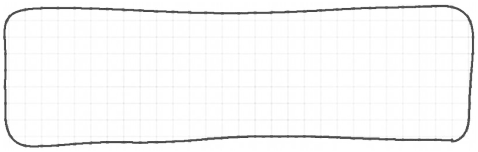
Problem: Wie die richtigen Hyperparameter finden?

1) Datensatz teilen in *Train* / *Validation* / *Test*.
2) Netz so optimieren, dass es *Train* möglichst gut *repräsentiert*.
3) Netz regularisieren, so dass es *Validation* möglichst gut *vorhersagt*.
4) Fertiges Netz mit (ganz neuem) *Test*-Datensatz überprüfen.

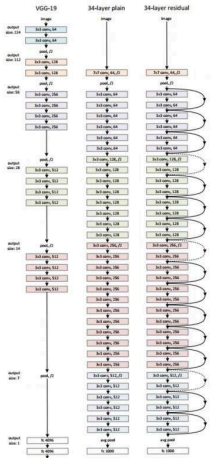


5.3 Regularisierung

7 goldene Regeln

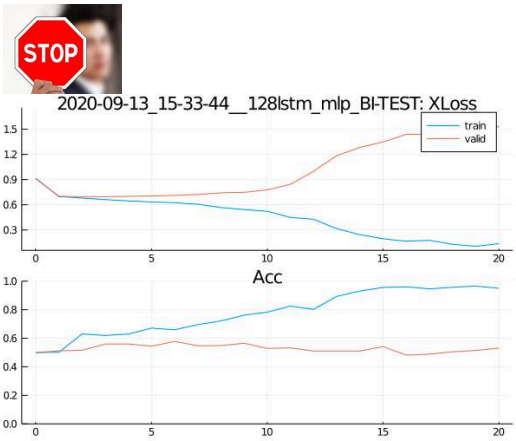


Keep it knee-high to a grasshopper!



5.4 early Stopping!

Aufhören, sobald der Loss der Valisierungsdaten wieder steigt:



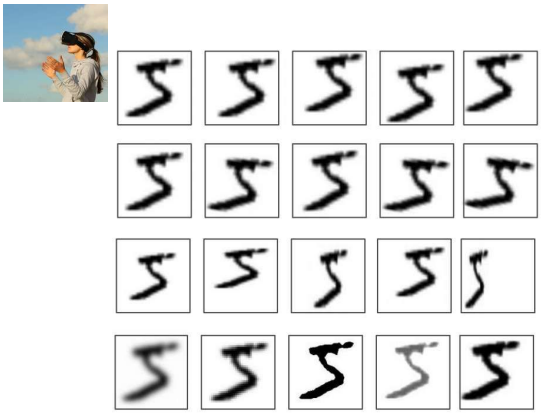
5.5 Be greedy!

Je mehr Trainingsdaten, desto besser! (das gilt auch, wenn man schon viele Daten hat)

1. Keep it knee-high to a grasshopper: simple is better than complicated!
2. Be anxoius: Early stopping is beautiful!
3. Be greedy: more is better than much!
4. Become rich: augment the world!
5. Have fun: go to Carnival in Venice!
6. Forget it all: weight decay!
7. Join the Emmentaler conspiracy: drop 'em out!

5.6 Augmentation!

Alle Modifikationen erzeigen, die sinnvoll sind:



5.7 Carneval in Venedig!

Rauschen und maskieren der Daten hilft:

5.8 Weight Decay!

Das originale Weight-Decay von Paul Werbos:

Biologisch Plausibel: Die Natur verringert Gewichte nicht durch Training, sondern durch Zeit (= nicht genutzte Axone verkümmern).

$$\Delta_p w_{ij}(t) = \eta \cdot o_{pi} \cdot \delta_{pj} - d \cdot w_{ij}(t-1)$$

Umsetzung:

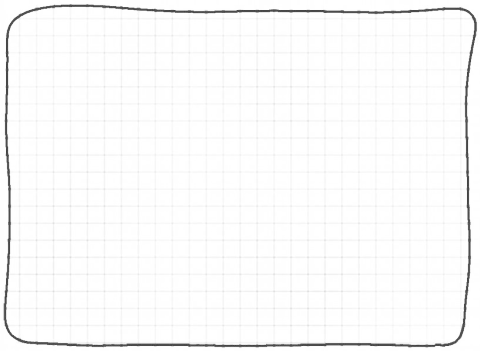
- Typische Werte: 0,000001 < d < 0,01
- Damit ist sogar Training mit der originalen Perzeptronlernregel möglich.

Weight-Decay durch L2-Regularisierung:

Begrenzung der Gewichte durch Loss-Manipulation

$$\mathcal{L}^{L2}(W) = \mathcal{L}(W) + \frac{d}{2n}\sum_i w_{ij}^2$$

$$w_{ij}(t+1) = w(t) + \Delta w_{ij} = w(t) - \eta\frac{\partial \mathcal{L}(W)}{\partial w_{ij}}$$

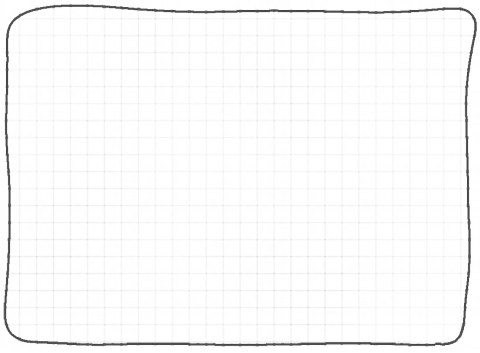


Weight-Decay durch L1-Regularisierung:

Begrenzung der Gewichte durch Loss-Manipulation

$$\mathcal{L}^{L1}(W) = \mathcal{L}(W) + \frac{d}{n}\sum_i |w_{ij}|$$

$$w_{ij}(t+1) = w(t) + \Delta w_{ij} = w(t) - \eta\frac{\partial \mathcal{L}(W)}{\partial w_{ij}}$$



- Original:

$$\Delta_p w_{ij}\left(t\right) = \eta \cdot o_{pi} \cdot \delta_{pj} - d \cdot w_{ij}\left(t-1\right)$$

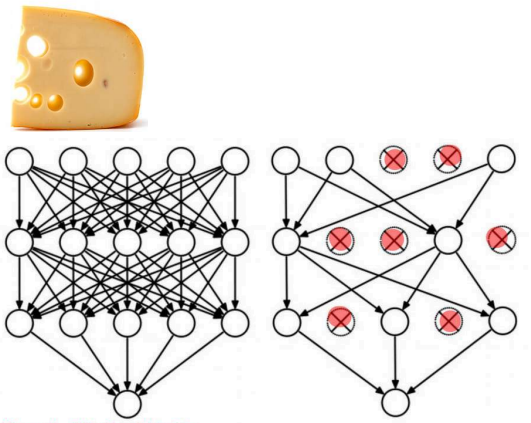
- L2:

$$w_{ij}(t+1) = (1 - \frac{\eta \cdot d}{n}) \cdot w_{ij}(t) - \eta \frac{\partial \mathcal{L}(W)}{\partial w_{ij}}$$

- L1:

$$w_{ij}(t+1) = w_{ij}(t) - \frac{\eta \cdot d}{n} \text{sgn}(w_{ij}) - \eta \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

5.9 Drop-Outs!



2010-14: Geoff Hinton [arXive 2012]

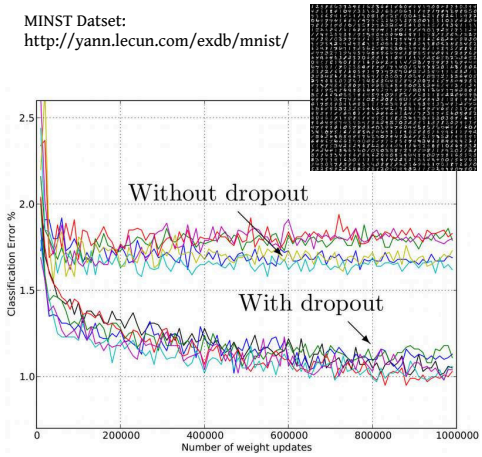
Journal of Machine Learning Research 15 (2014) 1929-1958 Submitted 11/13; Published 6/14

Dropout: A Simple Way to Prevent Neural Networks from Overfitting

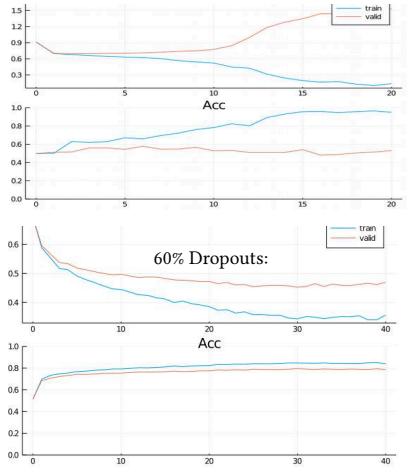
Nitish Srivastava NITISH@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Alex Krizhevsky KRIZ@CS.TORONTO.EDU
Ilya Sutskever ILYA@CS.TORONTO.EDU
Ruslan Salakhutdinov RSALAKHU@CS.TORONTO.EDU
Department of Computer Science
University of Toronto
10 Kings College Road, Rm 3302
Toronto. Ontario. M5S 3G4. Canada.

A motivation for dropout comes from a theory of the role of sex in evolution (Livnat et al., 2010). Sexual reproduction involves taking half the genes of one parent and half of the other, adding a very small amount of random mutation, and combining them to produce an offspring. The asexual alternative is to create an offspring with a slightly mutated copy of the parent's genes. It seems plausible that asexual reproduction should be a better way to optimize individual fitness because a good set of genes that have come to work well together can be passed on directly to the offspring. On the other hand, sexual reproduction is likely to break up these co-adapted sets of genes, especially if these sets are large and, intuitively, this should decrease the fitness of organisms that have already evolved complicated co-adaptations. However, sexual reproduction is the way most advanced organisms evolved.

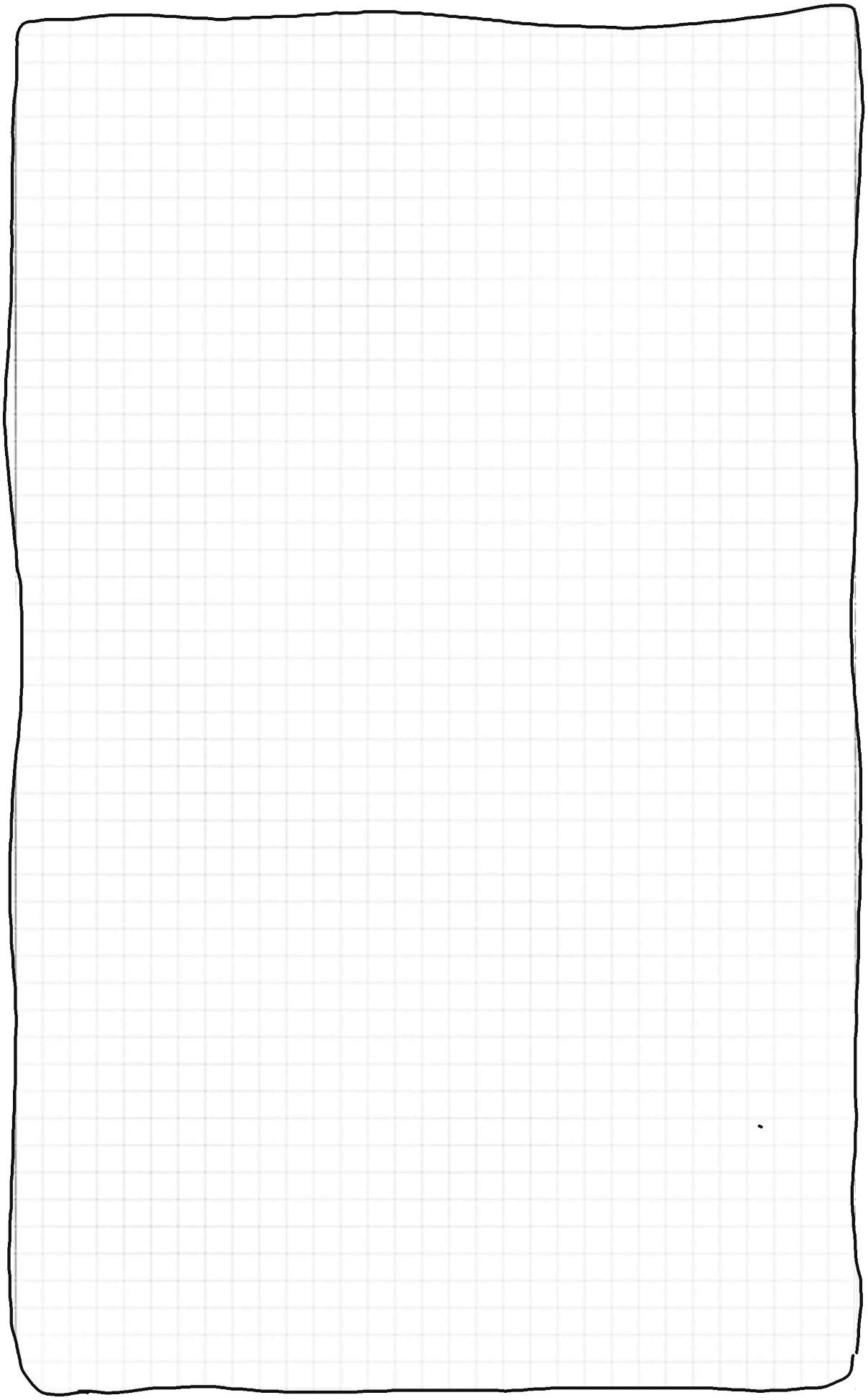
Dropouts am MNIST-Datensatz:



Dropouts am RNN:



5.10 Regularisierung ist einfach alles!

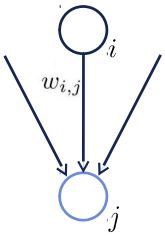


Kapitel 6:

Slipping Jimmy - Tipps und Tricks am Perceptron

6.1 Ein Wort zu Aktivierungsfungionen

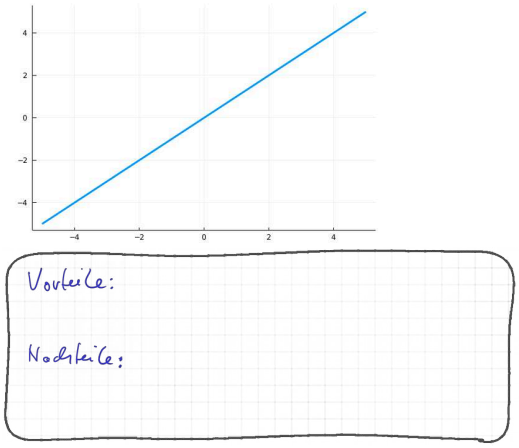
Das Neuron:



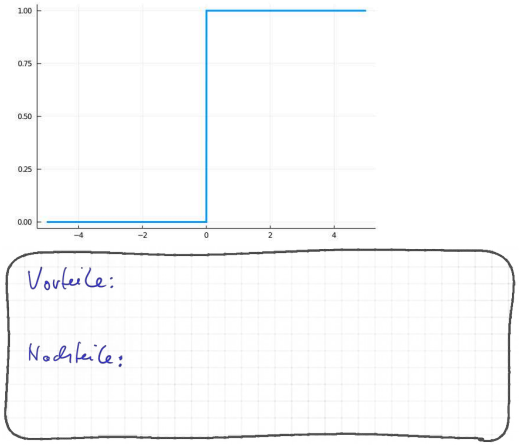
$$\text{net}_j = f(o_{i..n}, w_{i..n,j}) = \sum_i o_i \cdot w_{ij}$$

$$a_j(t+1) = f_{act}(net_j(t), \theta_j)$$

Idendität:

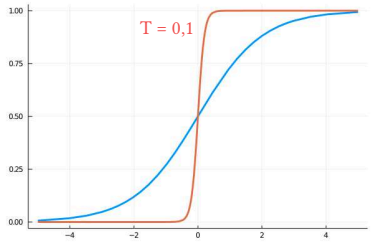


Schwellenwert:



Sigmoide Funktionen

Logistic (aka sigmoid):



„Die Mutter aller Sättigungsfunktionen“

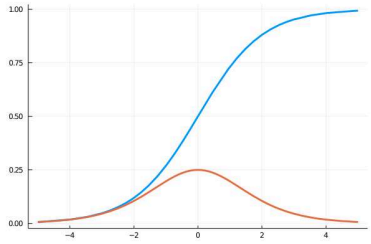
$$f(x) = \frac{1}{1+e^{-x}}$$

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

Vorteile:

Nachteile:

Gradient der Logistic-Fun:



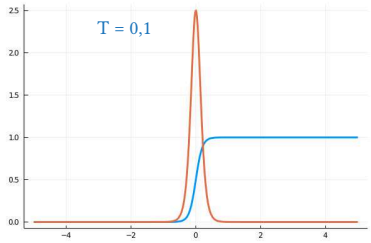
$$f(x) = \frac{1}{1+e^{-x}}$$

```
julia> f(x) = 1/(1+exp(-x))
julia> using AutoGrad
julia> plot([f, grad(f)])
```

Vorteile:

Nachteile:

Gradient der Logistic-Fun mit T=0,1:



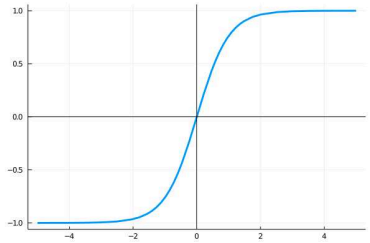
$$f(x) = \frac{1}{1+e^{-\frac{x}{T}}}$$

```
julia> f(x) = 1/(1+exp(-x/0.1))
julia> using AutoGrad
julia> plot([f, grad(f)])
```

Vorteile:

Nachteile:

tanh:



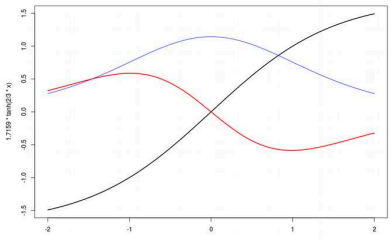
$$f(x) = \frac{2}{1 + e^{-x}} - 1$$

$$f(x) = \tanh(x)$$

Vorteile:

Nachteile:

Magischer tanh von Yann LeCun

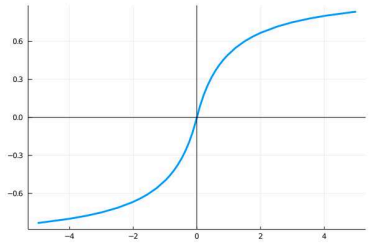


$$-\frac{e^{\pi-1}\cos(e\pi)}{\pi} \approx 1,7159$$

$$f^{tanh}(net) = 1,7159 \cdot \tanh(\frac{2}{3}net)$$

maximale Änderung des Gradienten bei +1.0 und -1.0

Softsign:



schönerer Gradient

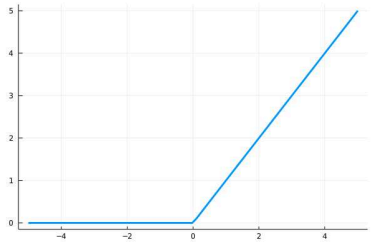
$$f(x) = \frac{x}{1 + |x|}$$

Vorteile:

Nachteile:

Rectified linear Units

ReLU:



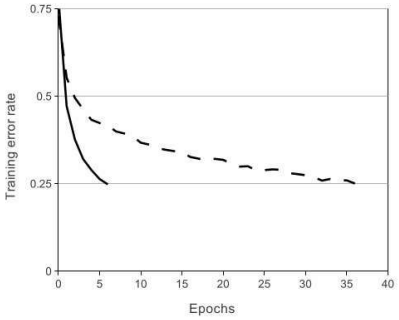
```
julia> relu(x) = max(0,x)
```

Vorteile:

Nachteile:

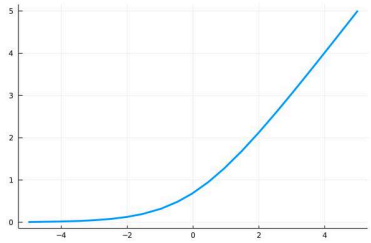
... trainiert schnell!

Figure 1: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25%training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line).



Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. ImageNet Classification with Deep Convolutional Neural Networks, in: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (Eds.), Advances in Neural Information Processing Systems 25. Curran Associates, Inc., pp. 1097–1105.

SoftPlus:



$$f(x) = \ln(1 + e^{-x})$$

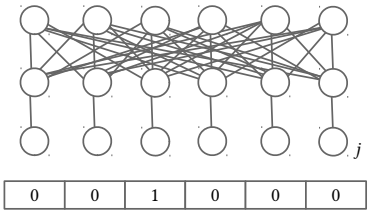
Vorteile:

Nachteile:

SoftMax

Bei Klassifizierung gegen einen One-Hot-Vektor:

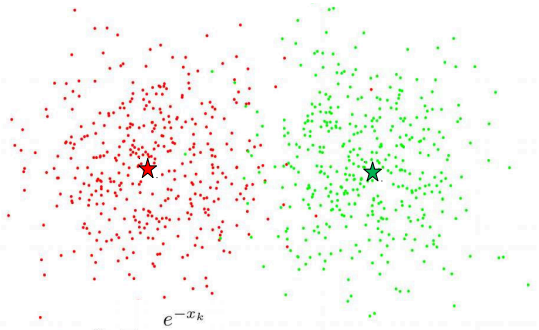
$$o_j = \frac{e^{x_j}}{\sum_k e^{x_k}}$$



→ Maximum Likelihood Estimation

→ Softmax Classifier

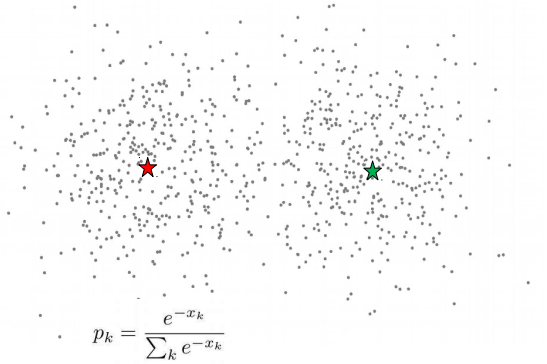
Gedankenexperiment Reis:



$$p_k = \frac{e^{-x_k}}{\sum_k e^{-x_k}}$$

$$p_{grün} = \frac{e^{-x_{grün}}}{e^{-x_{rot}} + e^{-x_{grün}}}$$

Wahrscheinlichkeiten:

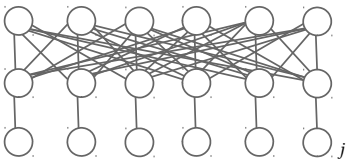


$$p_k = \frac{e^{-x_k}}{\sum_k e^{-x_k}}$$

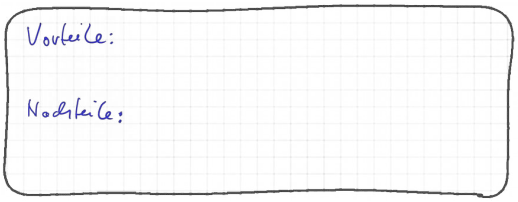
$$p_{grün} = \frac{e^{-x_{grün}}}{e^{-x_{rot}} + e^{-x_{grün}}}$$

SoftMax Likelihoodschätzer:

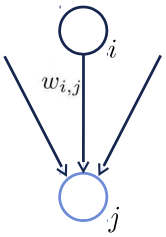
Sigmoide multinomiale logistische Regression



$$o_j = \frac{e^{x_j}}{\sum_k e^{x_k}}$$



Zusammenfassung:



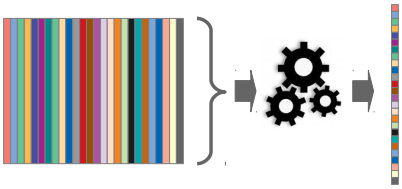
- linear
- logistic/*sigmoid*
- tanh
- softsign
- *magischer* tanh
- ReLU
- ReLu-ähnliche
- Softmax

$$\text{net}_j = f(o_{i..n}, w_{i..n,j}) = \sum_i o_i \cdot w_{ij}$$

$$a_j(t+1) = f_{act}(net_j(t), \theta_j)$$

6.2 Ein Wort zu Minibatches

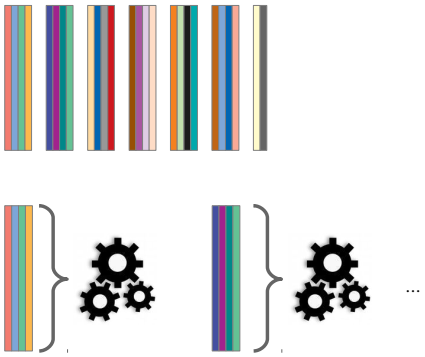
Die Theorie:



... geht bei kleinen Datensätzen!

aber was, wenn 1 000 000?

Eine Minibatch im Tensor:



Wie verhält sich das Training bei Minibatches unterschiedlicher Größe?

- GPU-Server `Yamata no Orochi`, MNIST-Dataset und großes MLP (um den Effekt zeigen zu können):

```
using Base.Iterators: flatten
using IterTools: ncycle, takenth
using Statistics: mean
using MLDatasets: MNIST
import Knet # load, save
using Knet: KnetArray, nll, zeroone, progress!, sgd,
            param, param0, dropout, relu, minibatch, Data
```

```
xtrn,ytrn = MNIST.traindata(Float32); ytrn[ytrn.==0] .= 10
xtst,ytst = MNIST.testdata(Float32);  ytst[ytst.==0] .= 10
```

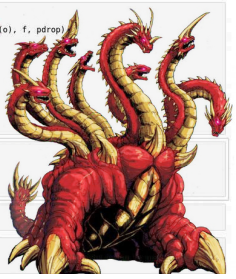
```
...
```

```
struct Dense; w; b; f; p; end
(d::Dense)(x) = d.f.(d.w * dropout(x,d.p) .+ d.b)
Dense(i::Int, o::Int,f=relu; pdrop=0) = Dense(param(o,i), param0(o), f, pdrop)

flat(x) = mat(x)

struct MLP
    layers
    MLP(layers...) = new(layers)
end
(c::MLP)(x) = (for l in c.layers; x = l(x); end; x)
(c::MLP)(x,y) = nll(c(x),y)
(c::MLP)(d::Data) = mean(c(x,y) for (x,y) in d)
```

```
mlpnet =   MLP(flat,
               Dense(784,512),
               Dense(512,256,pdrop=0.3),
               Dense(256, 64),
               Dense( 64, 10,identity))
```

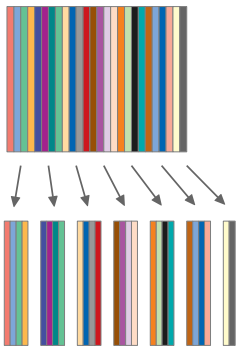
```
dtrn = minibatch(xtrn, ytrn, 1; xsize = (28,28,1,:))
(x,y) = first(dtrn)
```

```
dtrn = minibatch(xtrn, ytrn, ; xsize = (28,28,1,:))
progress!(sgd(mlpnet, ncycle(dtrn, 10)))
GC.gc(true)
```



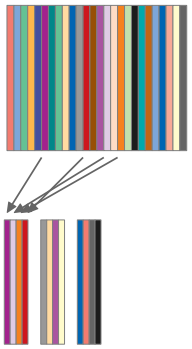
Erzeugen der MBs:

Systematisch oder zufällig:



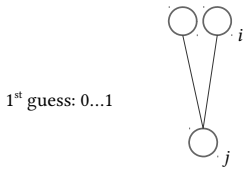
Systematisch oder zufällig:

Systematisch oder zufällig:

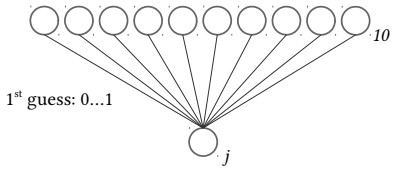


6.3 Ein Wort zur Initialisierung

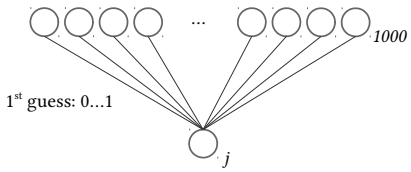
Wie groß ist der erwartete Netinput?



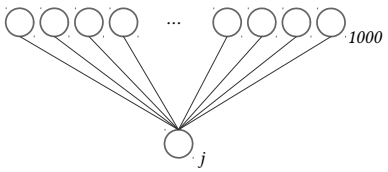
Wie groß ist der erwartete Netinput?



Wie groß ist der erwartete Netinput?

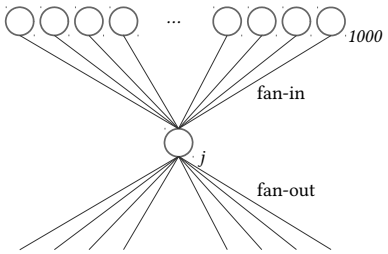


Normieren der Gewichte:



$$\frac{\text{rand}()}{\sqrt{\text{fan-in}}}$$

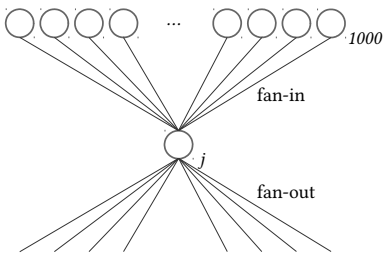
Xavier:



$$\text{uniform}(-1 \cdots +1) \cdot \frac{\sqrt{6}}{\sqrt{\text{fan-in} \cdot \text{fan-out}}}$$

Xavier Glorot, Yoshua Bengio:
Proceedings of the 13th International Conference
on Artificial Intelligence and Statistics (AISTATS) 2010

He:



$$\text{normal}() \cdot \frac{\sqrt{2}}{\sqrt{\text{fan-in}}}$$

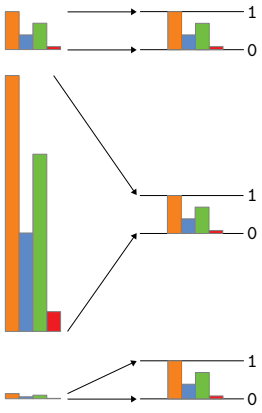
Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research, arXiv:1502.01852v1 [cs.CV] 6 Feb 2015

Zusammenfassung:

- einfach:
  - uniform(0,1)
  - uniform(-1,1)
  - normal()
- besser:
  - skaliert mit sqrt(fan-out)
- optimiert für sigmoide Actfun (logistic, tanh, softsign):
  - Xavier
- optimiert für ReLU:
  - Kaiming

6.4 Ein Wort zur Normalisierung

Normalisierung des Inputs:



möglich sind:
- min-max 0-1
- min-max -1 - +1
- z-score

`predict()` nicht vergessen!

Batchnormalisierung:

- Nicht über den gesamten Datensatz sondern batch-weise
- Innerhalb des Netzes (als Layer) möglich
- μ und σ können optimiert werden!

→ sorgt für Stabilität (ReLU)

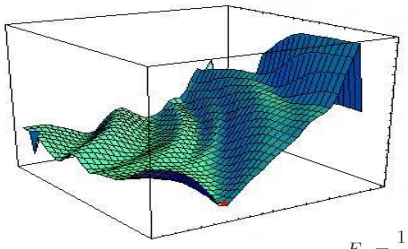
→ sorgt für Rauschen

wichtig/Vorsicht:

- μ und σ beim Test anwenden!
- batchnorm ist teuer!

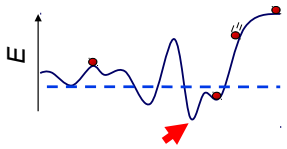
6.5 Ein Wort zu Flat Spots

So stellen wir uns die Fehlerfläche vor:

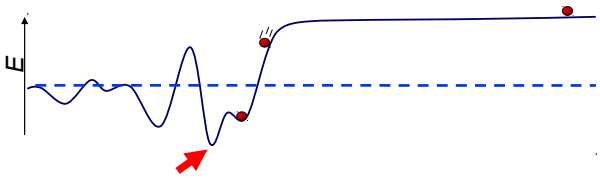


$$E_p = \frac{1}{2} \cdot \sum_j (t_{pj} - o_{pj})^2$$

$$E = \sum_p E_p$$

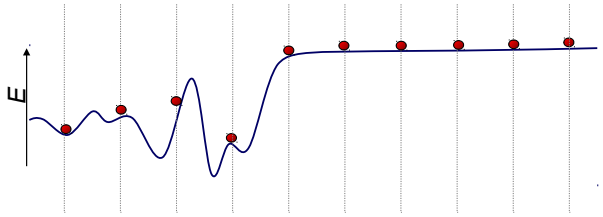


So sieht sie eher aus:



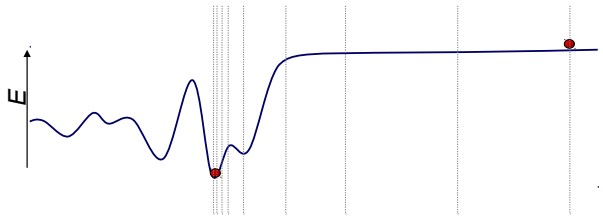
Flat Spots und kleine Lernrate:

Flat Spots und große Lernrate:

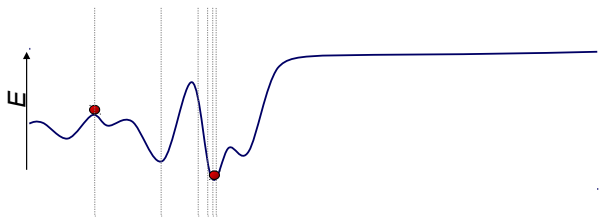


So ist es gut!

- Entkommen aus Flat Spots
- Überspringen von lokalen Minima

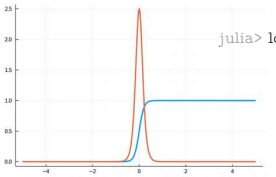


- Entkommen aus Flat Spots
- Überspringen von lokalen Minima



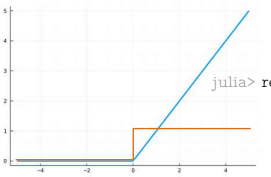
Ursache der Flat Spots

- Gradientenproblem sigmoider Funktionen



julia> logistic(x) = 1/(1 + exp(-x))

- Gradientenproblem ReLU

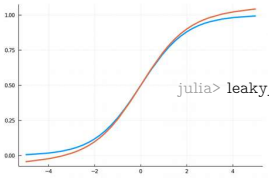


julia> relu(x) = max(0,x)

Flat-Spot-Elimination

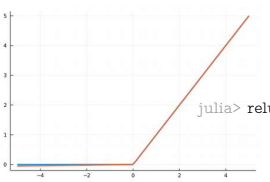
Flat-Spot-Elimination der Act-Funs:

- Gradientenproblem sigmoider Funktionen



julia> leaky_logistic(x) = 1/(1 + exp(-x)) + 0.01x

- Gradientenproblem ReLU



julia> relu(x) = max(0.01x, x)

Manhattan-Training

Das Problem der Flat-Spots entsteht, weil die Schrittgröße
mit der Steigung der Aktivierungsfunktion skaliert wird:

„Verbesserung“: Gradient weglassen: $\Delta w_{pij} = \eta \cdot o_{pi} \cdot \mathrm{sign}(\delta_{pj})$

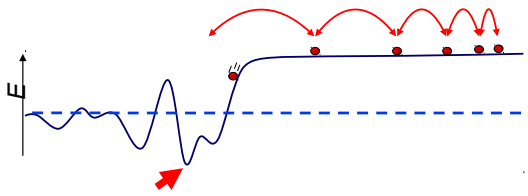
Funktioniert manchmal verblüffend gut!

Monentum-Term Training

Conjugate-Gradient-Training

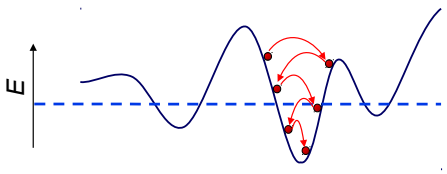
$$\Delta_p w_{ij}(t+1) = \eta\left((1-\alpha)\, o_{pi} \cdot \delta_{pj} + \alpha \cdot \Delta_p w_{ij}(t)\right)$$

α kann sehr groß sein (z.B. 0.9)



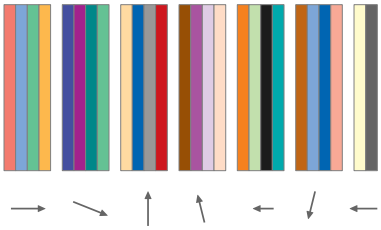
Conjugate-Gradient-Training

$$\Delta_p w_{ij}(t+1) = \eta\left((1-\alpha)\, o_{pi} \cdot \delta_{pj} + \alpha \cdot \Delta_p w_{ij}(t)\right)$$



Stochastic Grasient Descent:

- Jede Minibatch hat einen anderen Gradienten!

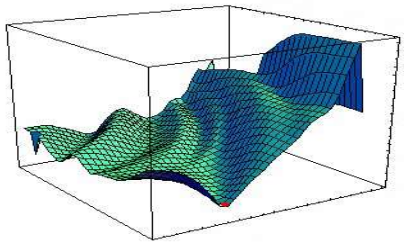


Zusammenfassung

Flat-Spot-Elimination:

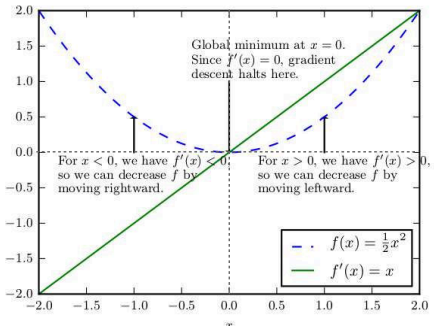
- Lernrate
- Act-Funs leaky
- Manhattan
- Momentum Term
- Minibatches

6.6 Optimierer



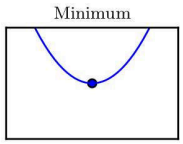
Parabelflächen:

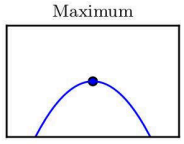
- passt!

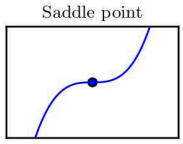


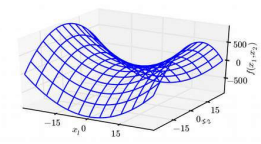
Flat Spots:

- Sattelpunkte sind ein Problem – je mehr Dimensionen je mehr Sattelpunkte
- lokale Minima

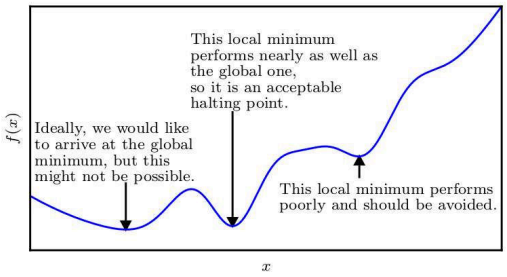




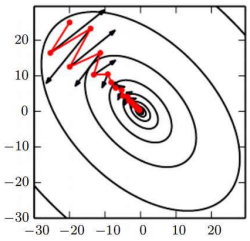




- Lokale Minima:

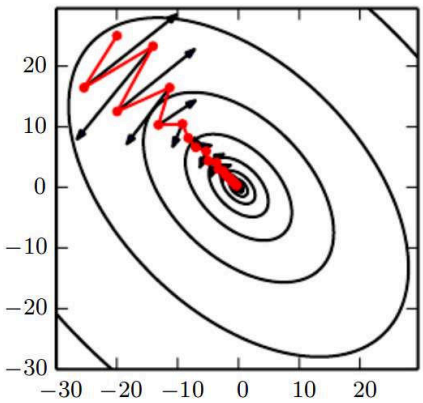


Steepest Descent folgt nicht dem Tal nach unten, sondern hat Fun in der Pipe.

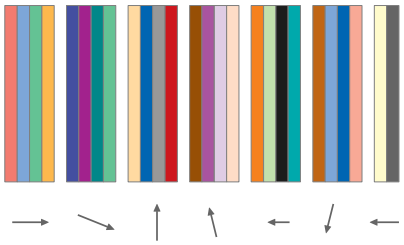


SGD

Momentum:



Minibatches:



Schwacher Optimierer (der Schwächste überhaupt!)

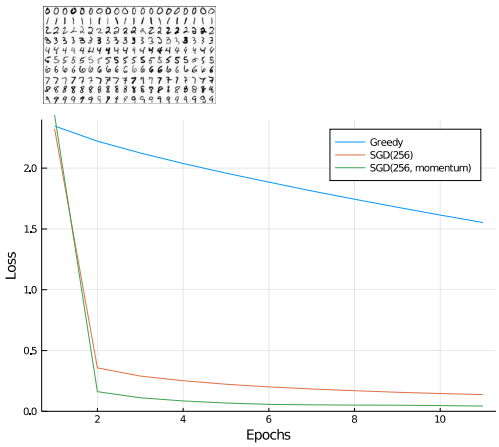
plus

Zufall (Noise)

ergibt

Superoptimierer, der lokale Minima überwindet.

Mit und ohne Minibatch:



Adaptive Optimierer

AdaGrad

Duchi, John, Elad Hazan, and Yoram Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, **12** (2011): 2121–2159.

$\boldsymbol{r} = 0;\ \delta = 10^{-7}$

$$\boldsymbol{g} = \frac{1}{n} \cdot \nabla_w \sum_{p=1}^{n} \mathcal{L}_p(\mathbf{o}, \mathbf{y})$$

$$\boldsymbol{r} = \boldsymbol{r} + \boldsymbol{g} \circ \boldsymbol{g}$$

$$\Delta \boldsymbol{w} = -\eta \frac{1}{\delta + \sqrt{r}} \circ \boldsymbol{g}$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta \boldsymbol{w}$$

```
g = grad(predict(w,x,y))
r .= r + g .^ 2
w .= w - η * g / (δ + √r)
```

RMSProp

Geoffey Hinton, unpublished, 2012

$\boldsymbol{r} = 0;\ \delta = 10^{-7};\ \rho < 1$

$$\boldsymbol{g} = \frac{1}{n} \cdot \nabla_w \sum_{p=1}^{n} \mathcal{L}_p(\mathbf{o}, \mathbf{y})$$

$$\boldsymbol{r} = \rho \boldsymbol{r} + (1-\rho) \boldsymbol{g} \circ \boldsymbol{g}$$

$$\Delta \boldsymbol{w} = -\eta \frac{1}{\sqrt{\delta + r}} \circ \boldsymbol{g}$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta \boldsymbol{w}$$

```
g = grad(predict(w,x,y))
r .= ρ * r + (1 - ρ) * g .^ 2
w .= w - η * g / √(δ + r)
```

Adam

Diederik P. Kingma, Jimmy Ba, Adam: A Method for Stochastic Optimization, 3rd International Conference for Learning Representations, San Diego, 2015.

$r = 0; m = 0; t = 0$

$\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$

$\eta = 0.001$

$$\boldsymbol{g} = \frac{1}{n} \cdot \nabla_w \sum_{p=1}^{n} \mathcal{L}_p(\mathbf{o}, \mathbf{y})$$

$$m = \beta_1 \cdot m + (1 - \beta_1) g$$

$$r = \beta_2 \cdot r + (1 - \beta_1) g \circ g$$

$$\hat{m} = \frac{m}{(1 - \beta_1^t)}$$

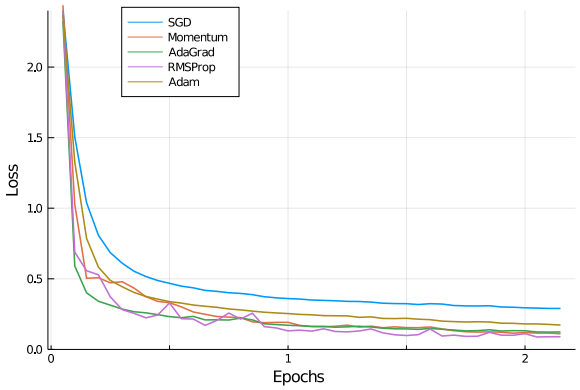
$$\hat{r} = \frac{r}{(1 - \beta_2^t)}$$

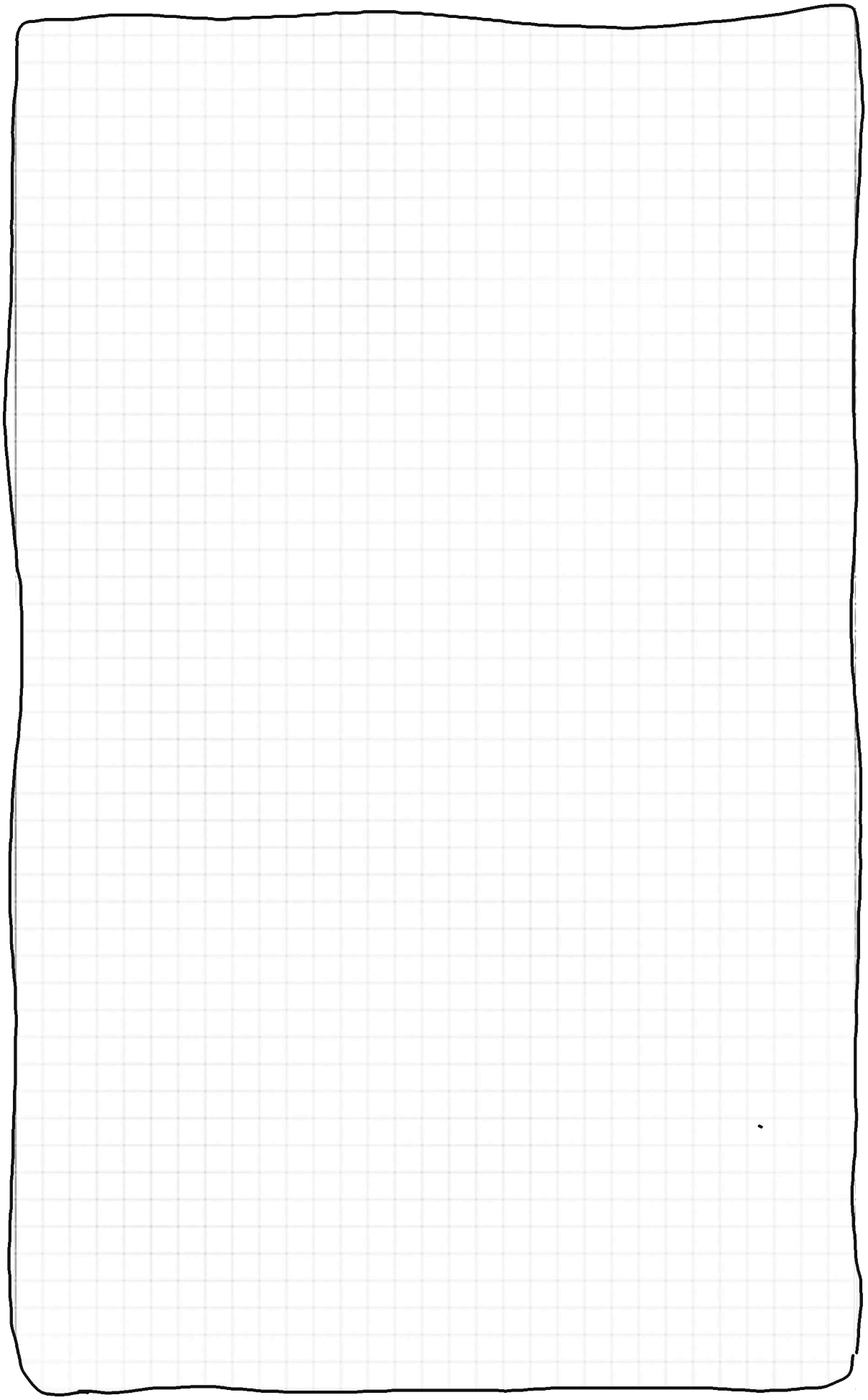
$$\Delta w = -\eta \cdot \frac{\hat{m}}{(\sqrt{\hat{r}} + \epsilon)}$$

$$\boldsymbol{w}(t+1) = \boldsymbol{w}(t) + \Delta \boldsymbol{w}$$

```
g = grad(predict(w,x,y))
m = β1 * m + (1-β1) * g
r =  β2 * r + (1-β2) * g .^ g
mhat = m ./ (1 - β1 ^ t)
mrhat = r ./ (1 - β2 ^ t)
w .= w - η * mhat / (√rhat + ε)
```

Vergleich der Optimierer:



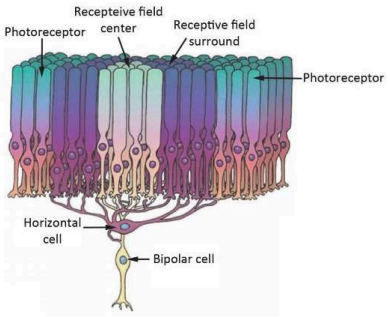


Kapitel 7:

Convolutional Neural Networks, CNN

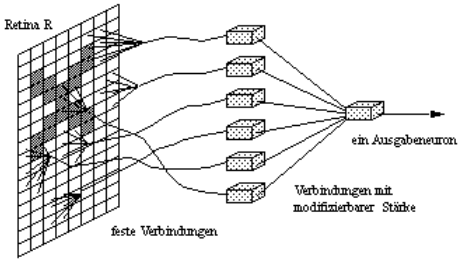
7.1 Idee und Motivation

Die Idee ist von der Biologie abgeschaut:



D. Nébouy, *Printing quality assessment by image processing and color prediction models.* Thesis, Université de Lyon, 2015.

und war bereits im ersten künstlichen Perceptron umgesetzt



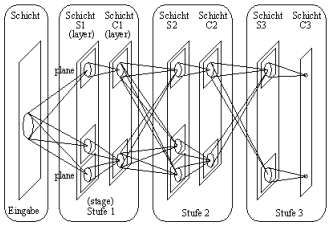
F. Rosenblatt, *„The perceptron: a probabilistic model for information storage and organization in the brain."*, Physiol. Rev., Bd. 65, Nr. 6, S. 386 – 408, 1958.

7.2 Neocognitron

Idee von Kunihiko Fokushima in den 1980-er Jahren:

S-Layer (simple):
Abb. des Sehfelds (z.B. 5x5)
meist gleiche Dimension wie Eingabe
gleicher Satz von Gewichten für alle Neurone

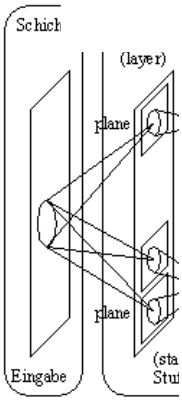
C-Layer (complex):
meist Zusammenfassung eines Bereichs
im S-Layer („sticht“ durch alle Layer einer Stufe)



Input: 19 x 19 x 1 S1: 19 x 19 x 12 S2: 21 x 21 x 38 S3: 13 x 13 x 35
C1: 21 x 21 x 8 C2: 13 x 13 x 19 C3: 7 x 7 x 23

Mehrere sog. Planes im S-Layer werden auf einzelne Muster trainiert

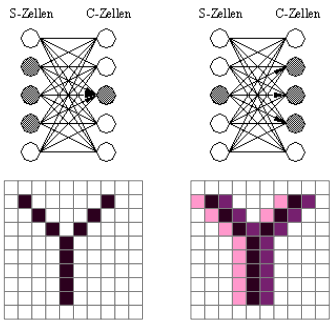
S-Layer (simple):
Abb. des Sehfelds (z.B. 5x5)
meist gleiche Dimension wie Eingabe
gleicher Satz von Gewichten für alle Neuron



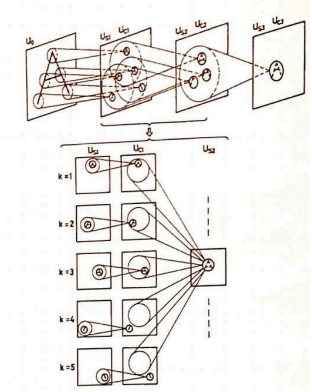
C-Layer sorgen für etwas Fuzzyness

Aktivierungsfunktion C-Layer: $c_j = \max(s_i)$

- Feuert, wenn mindestens eine Vorgängerzelle feuert



Mehrere S-C-Doppelschichten können bereits kompliziertere Strukturen erkennen, die aus den einfachen trainierten Mustern mehrerer Planes zusammengesetzt sind:



K. Fokushima, Analysis of the process of visual pattern recognition by the neocognitron. *Neural Networks* , **2** (1989) 413-420

Allerdings war das Neocognitrin mit der Hardware der 1980er Jahre nicht trainierbar!

- Jede S-Schicht wird mit einem vorgegebenen Muster trainiert.
- C-Schicht weight werden vorgegeben.
- Weight-Sharing innerhalb der Planes

Fazit:

- Neocognitron funktioniert erstaunlich gut.
- Mit 1980-er Hardware nicht machbar!

7.3 Das LeNet

LeNet-5: Der Urvater aller CNNs

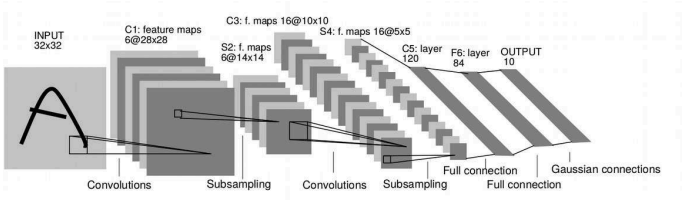
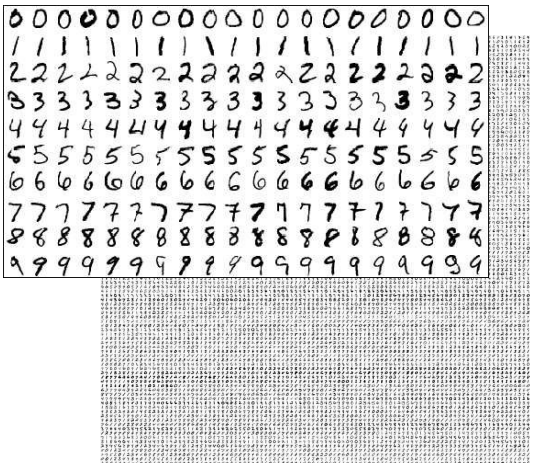


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner, 1998. Gradient-Based Learning Applied to Document Recognition.
Proc. IEEE 86, 2278–2324. http://yann.lecun.com/exdb/lenet/

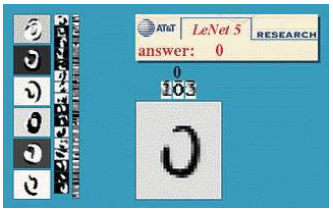
MNIST-Datensatz zur Evaluierung der Fähigkeiten zur Mustererkennung



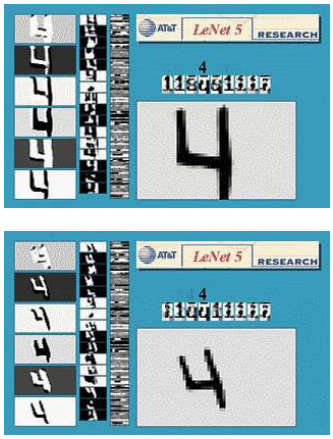
Modified National Institute of Standards and Technology

https://www.kaggle.com/c/digit-recognizer/

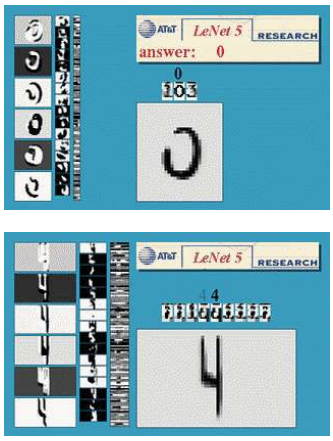
Yann LeCun



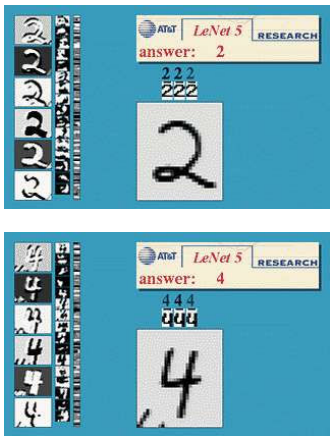
Skalierungsinvarianz und leichte Rotationsinvarianz



Translationsinvarianz und Stabilität geg. Deformation

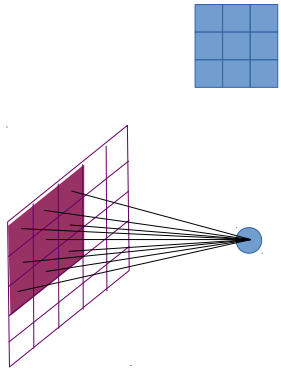


Rauschen stört erstaunlich wenig

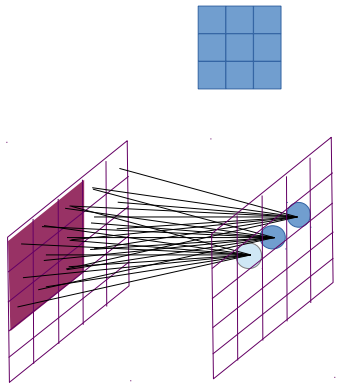


7.4 Funktionsweise eines CNN

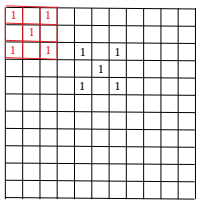
Simulation eines Rezeptiven Feldes als Filtermatrix

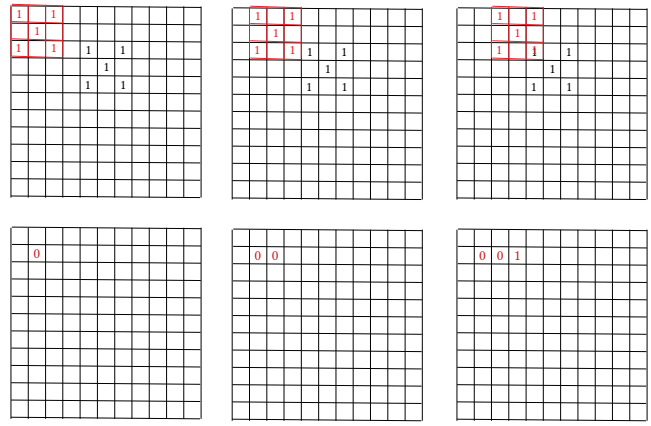


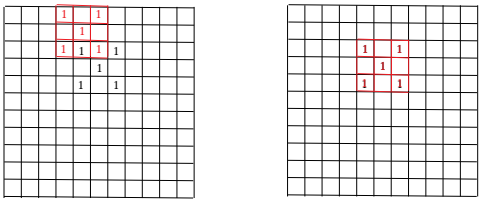
Die Faltung (Convolution) wird numerisch durchgeführt

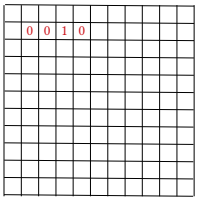


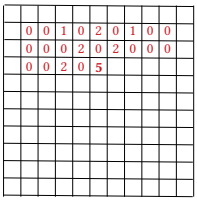
Anwendung eines einfachen Filters (X) auf ein Muster (X):

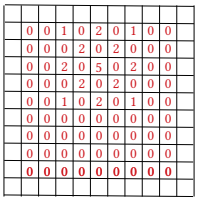




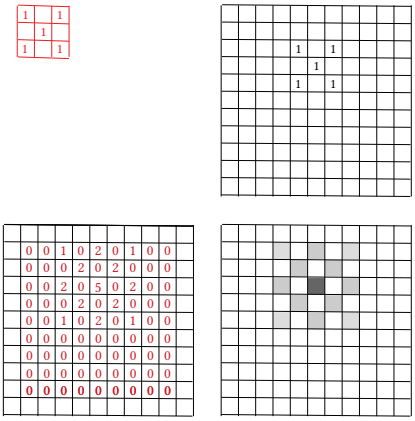




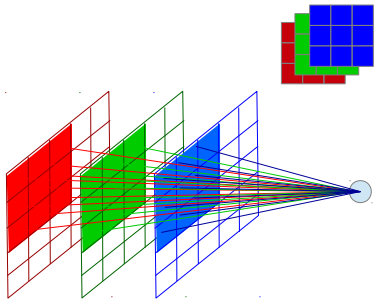




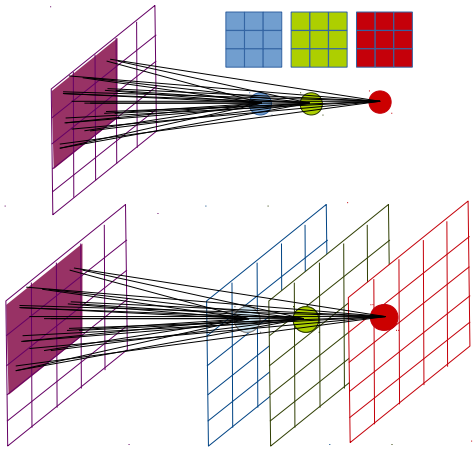
Nur an der Stelle mit perfektem Match gibt es ein maximales Signal!



Die Filtermatrix kann bei Bedarf auch 3-dimensional sein:



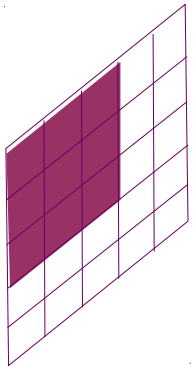
Und natürlich benötigt man viele Filter



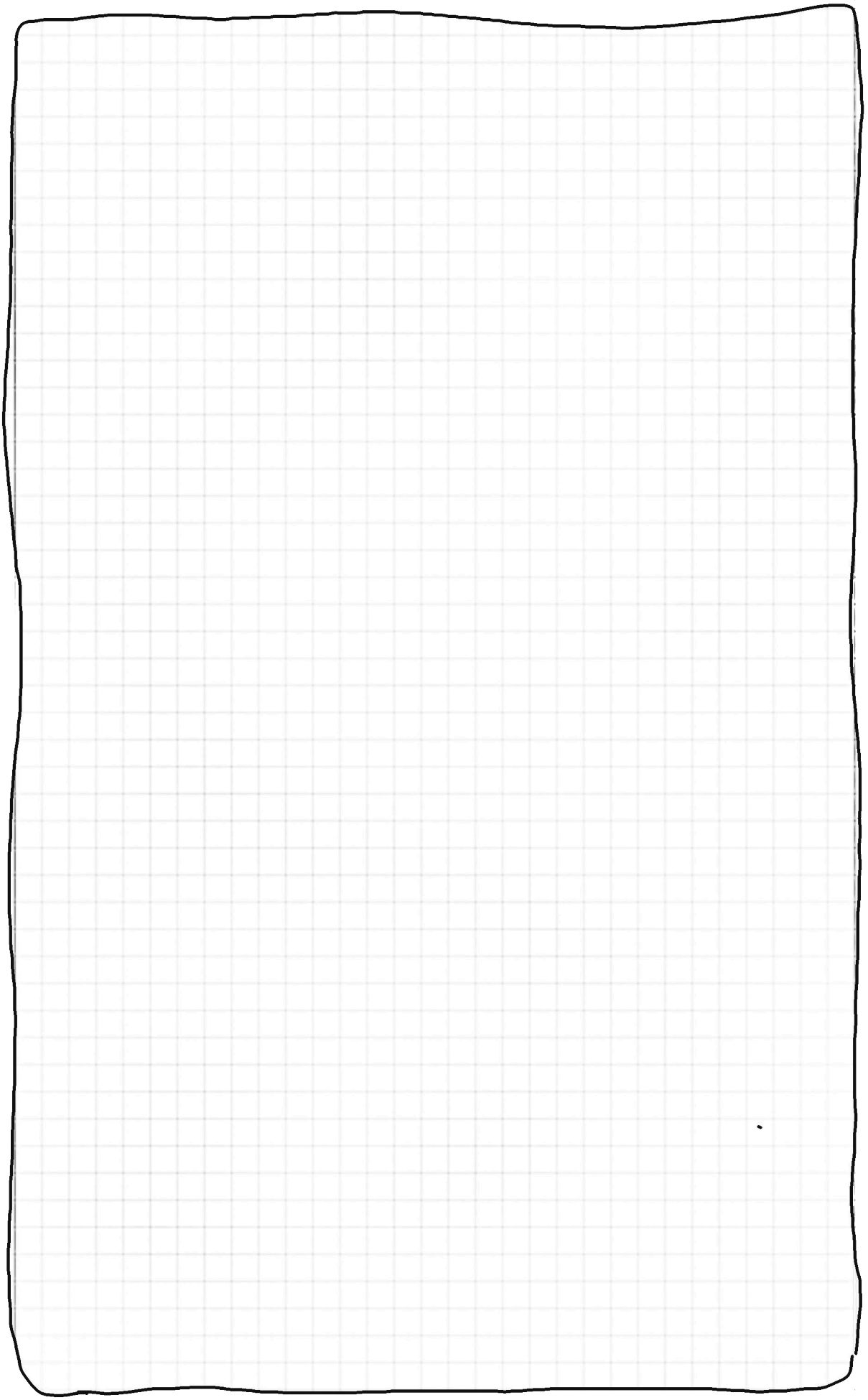
Aus den C-Layers des Neocognitron werden Pooling-Layer:

2 Möglichkeiten, die Zahl an Neuronen zu verkleinern:

- Feature Map kleiner (stride)
- Pooling (sum, max, av)

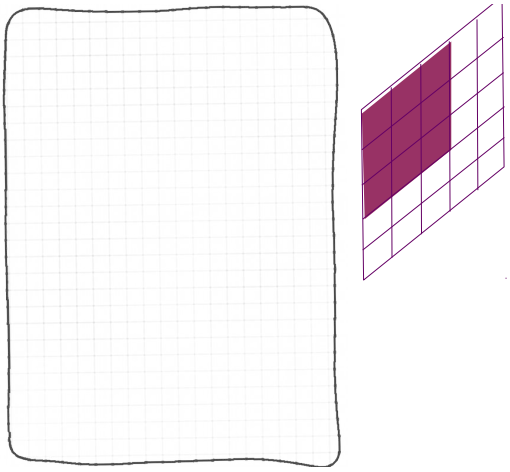


Convolution und Pooling

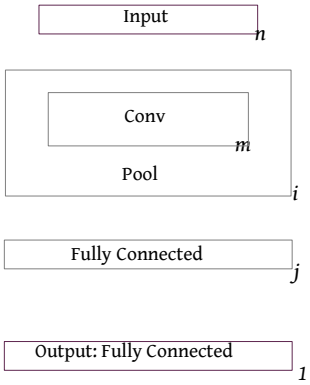


Einstellbare Parameter des Conv-Layers:

- Kernel
- Stride
- Padding
- Dilution
- Bias
- Actfun.



Generische Architektur des CNN:



Mögliche einfache Implementierung der Layer in Julia (mit den Helferlein-Funktionen von Knet):

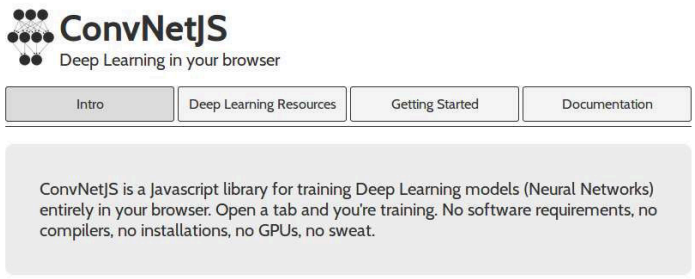
```
using Knet: conv4, pool, param, param0, relu

struct Conv
    w; b; f
    Conv(w1, w2, nm, nf, f=relu) = new(param(w1,w2,nm,nf), param0(1,1,nf,1), f)
end
(l::Conv)(x) = l.f.(conv4(l.w, x) .+ l.b)
l1 = Conv(5, 5, 3, 64)


struct Pool
    win
    Pool(win...) = new(win)
end
(l::Pool)(x) = pool(x, window=win)
l2 = Pool(2,2)
```

```
using Knet: dropout, batchnorm

struct Drop
    p
    Drop(p=0.0) = new(p)
end
(l::Drop)(x) = dropout(l.p)
l3 = Drop(0.4)


struct BatchNorm
    ε
    BatchNorm(ε=1e-5) = new(ε)
end
(l::BatchNorm)(x) = batchnorm(x, eps=l.ε)
l4 = BatchNorm()
```

```
# fully   edged:
#
struct Conv
    w
    b
    f
    p
    normalize
    Conv(w1, w2, nm, nf; actf=relu, drop=0.0, normalize=false) =
        new(param(w1,w2,nm,nf), param0(1,1,nf,1), actf, drop, normalize);
end

function (l::Conv)(x)
    x = l.f.(pool(conv4(l.w, x) .+ l.b))
    x = dropout(x, p)
    if l.normalize
        x = batchnorm(x)
    end
    return x
end

l5 = Conv(5, 5, 3, 64, normalize=true)
```

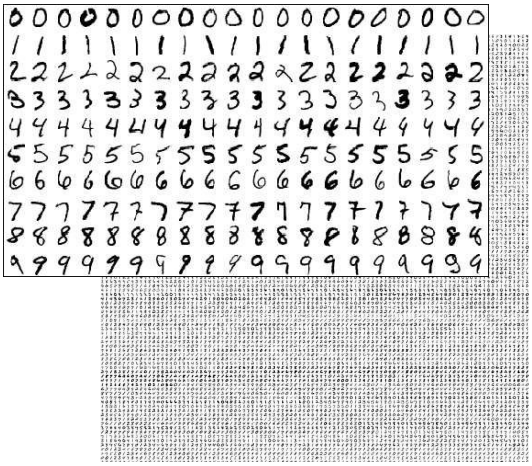
Spielwiese:

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html



7.5 Datensätze zum Test der CNNs

MNIST

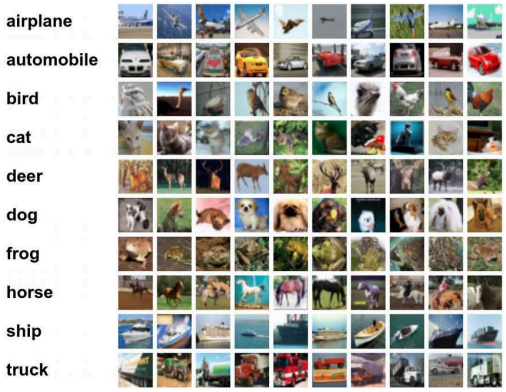


Modified National Institute of Standards and Technology

https://www.kaggle.com/c/digit-recognizer/

< 1% Error

CIFAR-10



ResNet: 6.5% Error

CIFAR-100

This dataset is just like the CIFAR-10, except it has 100 classes containing 600 images each. There are 500 training images and 100 testing images per class. The 100 classes in the CIFAR-100 are grouped into 20 superclasses.

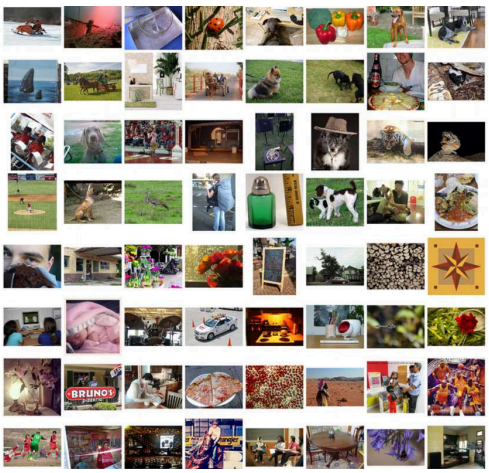
Superclass	Classes
aquatic mammals	*beaver, dolphin, otter, seal, whale*
fish	*aquarium fish, flatfish, ray, shark, trout*
flowers	*orchids, poppies, roses, sunflowers, tulips*
food containers	*bottles, bowls, cans, cups, plates*
fruit and vegetables	*apples, mushrooms, oranges, pears, sweet peppers*
household electrical devices	*clock, computer keyboard, lamp, telephone, television*
household furniture	*bed, chair, couch, table, wardrobe*
insects	*bee, beetle, butterfly, caterpillar, cockroach*
large carnivores	*bear, leopard, lion, tiger, wolf*
large man-made outdoor things	*bridge, castle, house, road, skyscraper*
large natural outdoor scenes	*cloud, forest, mountain, plain, sea*
large omnivores and herbivores	*camel, cattle, chimpanzee, elephant, kangaroo*
medium-sized mammals	*fox, porcupine, possum, raccoon, skunk*
non-insect invertebrates	*crab, lobster, snail, spider, worm*
people	*baby, boy, girl, man, woman*
reptiles	*crocodile, dinosaur, lizard, snake, turtle*
small mammals	*hamster, mouse, rabbit, shrew, squirrel*
trees	*maple, oak, palm, pine, willow*
vehicles 1	*bicycle, bus, motorcycle, pickup truck, train*
vehicles 2	*lawn-mower, rocket, streetcar, tank, tractor*

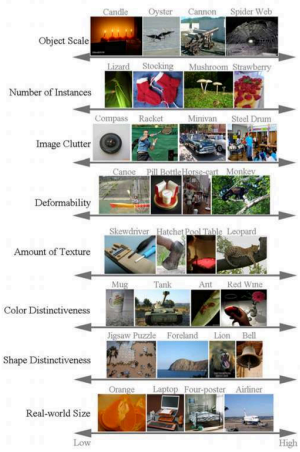
Yes, I know mushrooms aren't really fruit or vegetables and bears aren't really carnivores.

SVHN

600 000 Hausnummern von StreetView.

ImageNet





7.6 Die ImageNet-Challenge ILSVRC

ImageNet Large Scale Visual Recognition Challenge ILSVRC

ImageNet Large Scale Visual Recognition Challenge

The diversity of data in the ILSVRC image classification and single-object localization tasks. For each of the eight dimensions, we show example object categories along the range of that property.

The other properties were computed by asking human subjects to annotate each of the 1000 object categories.

Table 2 Scale of ILSVRC image classification task (minimum per class - maximum per class)

Year	Train images (per class)	Val images (per class)	Test images (per class)
Image classification annotations (1000 object classes)			
ILSVRC2010	1,261,406 (668–3047)	50,000 (50)	150,000 (150)
ILSVRC2011	1,229,413 (384–1300)	50,000 (50)	100,000 (100)
ILSVRC2012-14	1,281,167 (732–1300)	50,000 (50)	100,000 (100)

The numbers in parentheses correspond to (minimum per class–maximum per class). The 1000 classes change from year to year but are consistent between image classification and single-object localization tasks in the same year. All images from the image classification task may be used for single-object localization

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vis.* (2015) **115**, 211–252.

Die Anfänge der Challenge

Zu Beginn waren keine Neuronalen Netze am Start!
ILSVRC 2010

Codename	CLS	Insitutions
Hminmax	54.4	Massachusetts Institute of Technology
IBM	70.1	IBM research†, Georgia Tech‡
ISIL	44.6	Intelligent Systems and Informatics Lab., The University of Tokyo
ITNLP	78.7	Harbin Institute of Technology
LIG	60.7	Laboratoire d'Informatique de Grenoble
NEC	**28.2**	NEC Labs America†, University of Illinois at Urbana-Champaign‡, Rutgers∓
NII	74.2	National Institute of Informatics, Tokyo,Japan†, Hefei Normal Univ. Heifei, China‡
NTU	58.3	CeMNet, SCE, NTU, Singapore
Regularities	75.1	SRI International
UCI	46.6	University of California Irvine
XRCE	33.6	Xerox Research Centre Europe

Top-5 Error

Technologies:

1st (NEC) SIFT/SVM

2nd (XRCE) Fisher Vector Representation/PCA/SVM

ILSVRC 2011

Codename	CLS	LOC	Institutions
ISI	36.0	-	Intelligent Systems and Informatics lab, University of Tokyo
NII	50.5	-	National Institute of Informatics, Japan
UvA	31.0	**42.5**	University of Amsterdam†, University of Trento‡
XRCE	**25.8**	56.5	Xerox Research Centre Europe†, CIII‡

Technologies:

1st (XRCE) High-dimensional image signatures/SVM

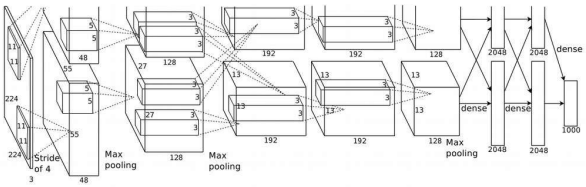
2nd (UvA) class-independent object hypothesis regions

AlexNet

Alex und Geoff

- Alex Krizhevsky
- Geoffrey Hinton

Aufbau des AlexNet



- Deeper
- Augmentation (shift/mirror ⟶ 2048x)
- Dropouts
- 2 GPUs
- Relu
- Batch-Normalisation

6d, 2 GPUs

Input	224x224x3 (Bilder RGB)	150528
Conv.	96 Feature Maps 11x11x3, Stride 4⟶55x55	290400
Max pooling	5x5 (überlappend, Stride 2x2)⟶27x27	69984
Conv.	256 Feature Maps, Kernel 5x5x48	186624
Max pooling	3x3 (überlappend, Stride 2x2)⟶13x13	43264
Conv.	384 Feature Maps, 3x3x256	64896
Conv.	384 Feature Maps, 3x3x192	64896
Conv.	256 Feature Maps, 3x3x192	43264
Max pooling	3x3 (überlappend, Stride 2x2)⟶6x6	9216
FC	4096	4096
FC	4096	4096
Output	1000 (Klassen)	1000

ILSVRC 2012

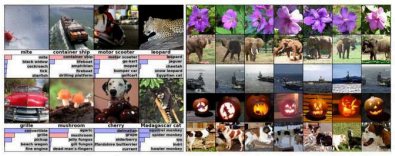
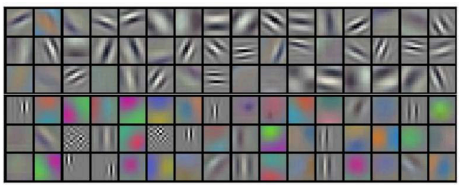


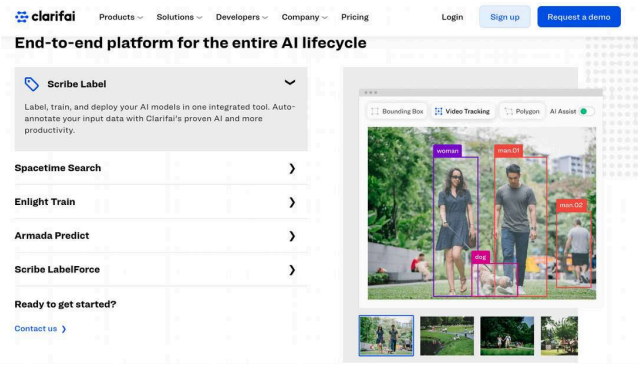
Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Top-5 Error: (test):	
SuperVision (AlexNet):	16,4%
ISI: (SIFT)	26,2%
OXFORD_VGG:	27,0%%
XRCE (SVM):	27,0%
Univ. Amsterdam (SIFT):	30,0%

Filter (11x11x3):



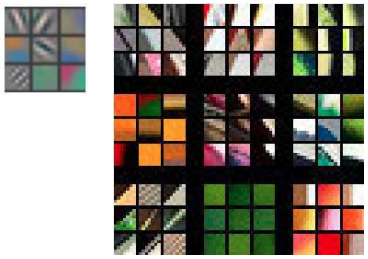
ZFNet



M.D. Zeiler, R. Fergus. *Visualizing and Understanding Convolutional Networks.* In: Computer Vision – *ECCV 2014.* ECCV 2014. Lecture Notes in Computer Science, vol 8689. Springer, Cham.

Visualisierung der Layer mit Deconvolution

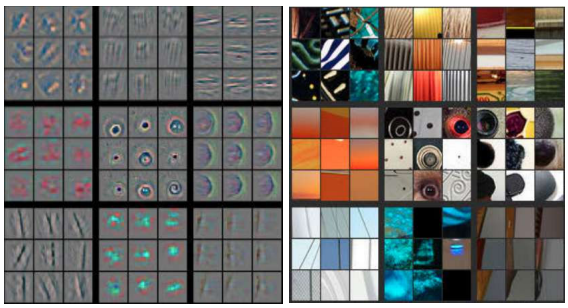
- Layer 1:



M.D. Zeiler, R. Fergus. *Visualizing and Understanding Convolutional Networks.* In: Computer Vision – *ECCV 2014.* ECCV 2014. Lecture Notes in Computer Science, vol 8689. Springer, Cham.

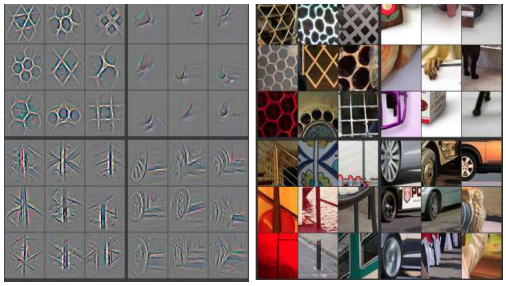
Visualisierung der Layer mit Deconvolution

- Layer 2:



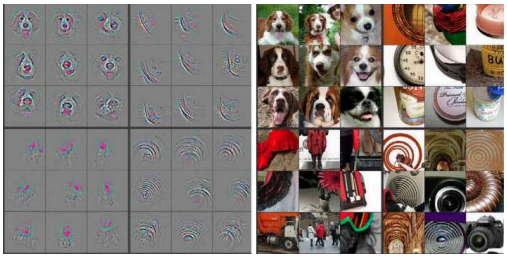
Visualisierung der Layer mit Deconvolution

- Layer 3:



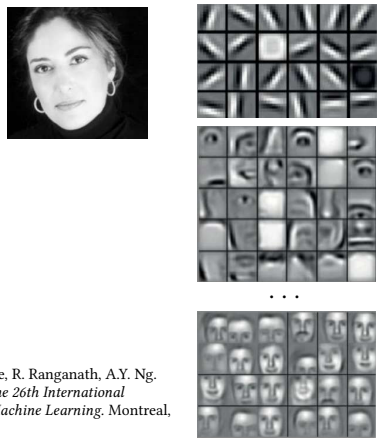
Visualisierung der Layer mit Deconvolution

- Layer 4:



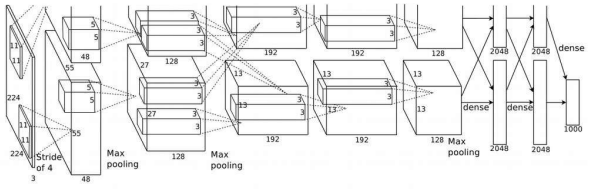
Visualisierung der Layer mit Deconvolution

- Layer 5:

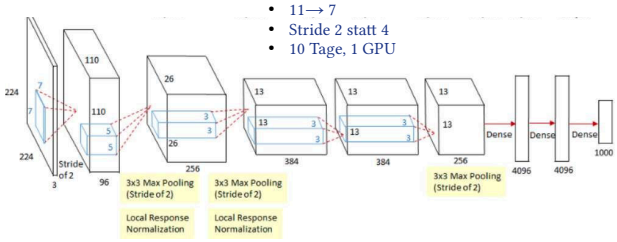


H. Lee, R. Grosse, R. Ranganath, A.Y. Ng. *Proceedings of the 26th International Conference on Machine Learning*. Montreal, Canada (2009).

Vergleich AlexNet und ZFNet



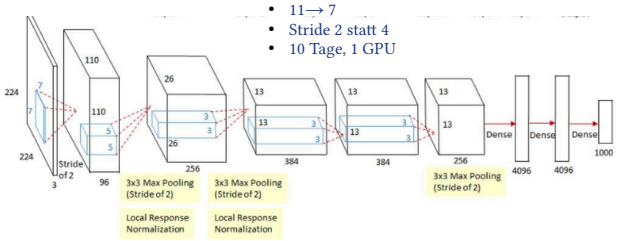
- 11⟶ 7
- Stride 2 statt 4
- 10 Tage, 1 GPU



ILSVRC 2013

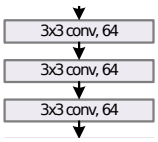
Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k

- 11⟶ 7
- Stride 2 statt 4
- 10 Tage, 1 GPU



Inception, GoogLeNet

Numerische Berechnung des Gradienten



$$\mathcal{F}^{l3}(\mathcal{F}^{l2}(\mathcal{F}^{l1}(x)))$$

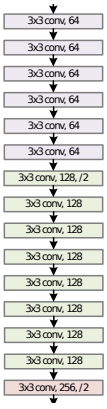
$$g^{l1} = \nabla \mathcal{F}^{l1}(x)$$

$$g^{l2} = \nabla \mathcal{F}^{l2}(x^{l1})$$

$$g^{l3} = \nabla \mathcal{F}^{l3}(x^{l2})$$

$$g^{tot} = g^{l1} \cdot g^{l2} \cdot g^{l3}$$

Vanishing Gradient



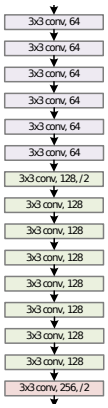
$$\mathcal{F}^{l3}(\mathcal{F}^{l2}(\mathcal{F}^{l1}(x)))$$

$$g^{tot} = g^{l1} \cdot g^{l2} \cdot g^{l3}$$

$$g^{tot} \approx 0.5^n$$

n = 3:	0.125
n = 5:	0.03125
n = 10:	0.000976562
n = 30:	9.31e-10
n = 60:	8.67e-19

Exploding Gradient



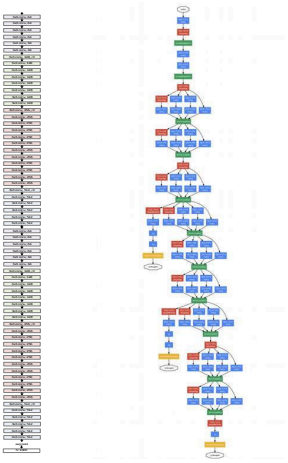
$$\mathcal{F}^{l3}(\mathcal{F}^{l2}(\mathcal{F}^{l1}(x)))$$

$$g^{tot} = g^{l1} \cdot g^{l2} \cdot g^{l3}$$

$$g^{tot} \approx 1.5^n$$

n = 3:	3.375
n = 5:	7.59375
n = 10:	57.665
n = 30:	191751.1
n = 60:	3.68e10

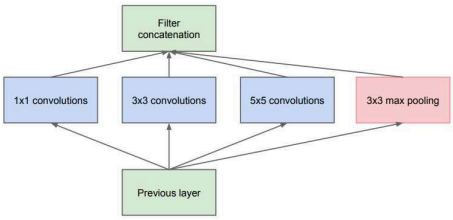
Inception



ILSVRC 2014

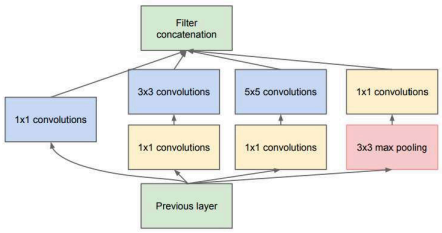
Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Einfaches Inception-Modul

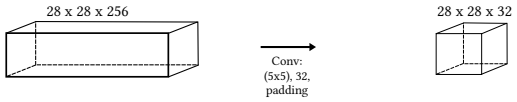


C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke und A. Rabinovich. Going Deeper with Convolutions. *CoRR* abs/1409.4842 (2014); https://arxiv.org/abs/1409.4842.

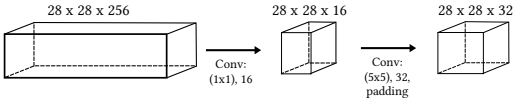
Inception-Modul mit Dimensionsreduktion



Der Trick mit 1x1-Convolution



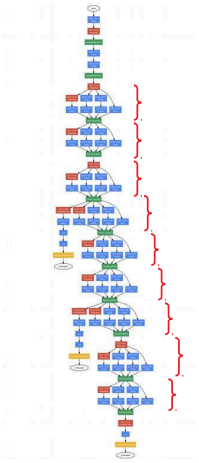
32 Filter mit: 5x5x256 weights
28x28 x 32 x 5x5x256 = **160 563 200 fpo**



16 Filter mit: 1x1x256 weights
28x28 x 16 x 1x1x256 = **3 211 264 fpo**

32 Filter mit: 5x5x16 weights
28x28 x 32 x 5x5x16 = **10 035 200 fpo**

Inception-Architektur

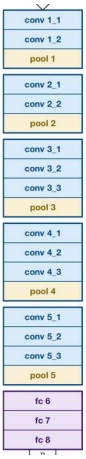


VGG Net

Architektur

- kleiner Filter (3x3)
- weniger Pooling

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					



ILSVRC 2014

- Visual Geometry Group, Oxford

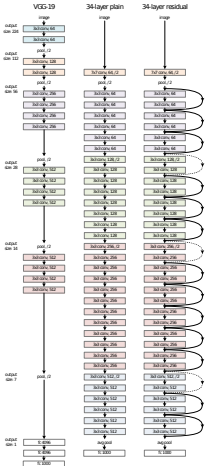
Table 5: **ConvNet evaluation techniques comparison.** In all experiments the training scale S v sampled from $[256; 512]$, and three test scales Q were considered: $\{256, 384, 512\}$.

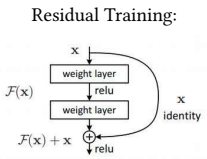
ConvNet config. (Table 1)	Evaluation method	top-1 val. error (%)	top-5 val. error (%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	**24.4**	**7.2**
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

K. Simonyan, A. Zisserman.
Very Deep Convolutional Networks for Large-Scale Image Recognition, ICLR 2015

ResNet

Residual Training





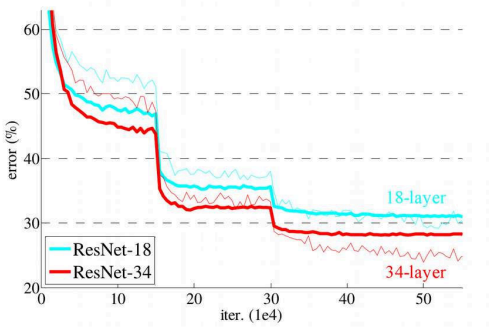
ILSVRC 2015

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

K. He, X. Zhang, S. Ren und J. Sun. *Deep Residual Learning for Image Recognition*. CoRR abs/1512.03385 (2015).

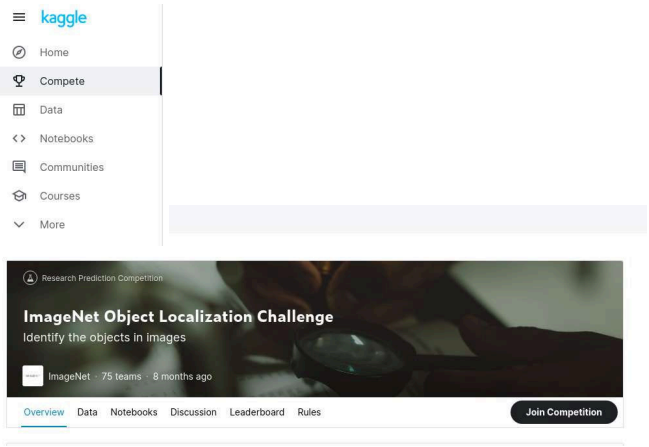
ResNet Training



Training on ImageNet. Thin curves denote training error, and bold curves denote validation error of the center crops. ResNets of 18 and 34 layers.

ILSVRC und Kaggle

Seit 2018 bei Kaggle:



7.7 Modell Zoo

TensorFlow Model Garden https://github.com/tensorflow/models

Model Zoo
Discover open source deep learning code and pretrained models.
https://modelzoo.co/

PyTorch MODEL ZOO https://pytorch.org/serve/model_zoo.html

Open Neural Network Exchange
The open standard for machine learning interoperability
ONNX
https://onnx.ai/

Keras Sandbox

```
from tensor ow.keras.applications.vgg16 import VGG16
base_model = VGG16(input_shape=(224, 224, 3),
                   include_top=False, weights='imagenet')

from tensor ow.keras.applications.inception_v3 import InceptionV3
base_model = InceptionV3(input_shape = (150, 150, 3),
                         include_top = False, weights = 'imagenet')


from tensor ow.keras.applications import ResNet50
base_model = ResNet50(input_shape=(224, 224,3),
                      include_top=False, weights='imagenet')


for layer in base_model.layers:
    layer.trainable = False
```

ONNX

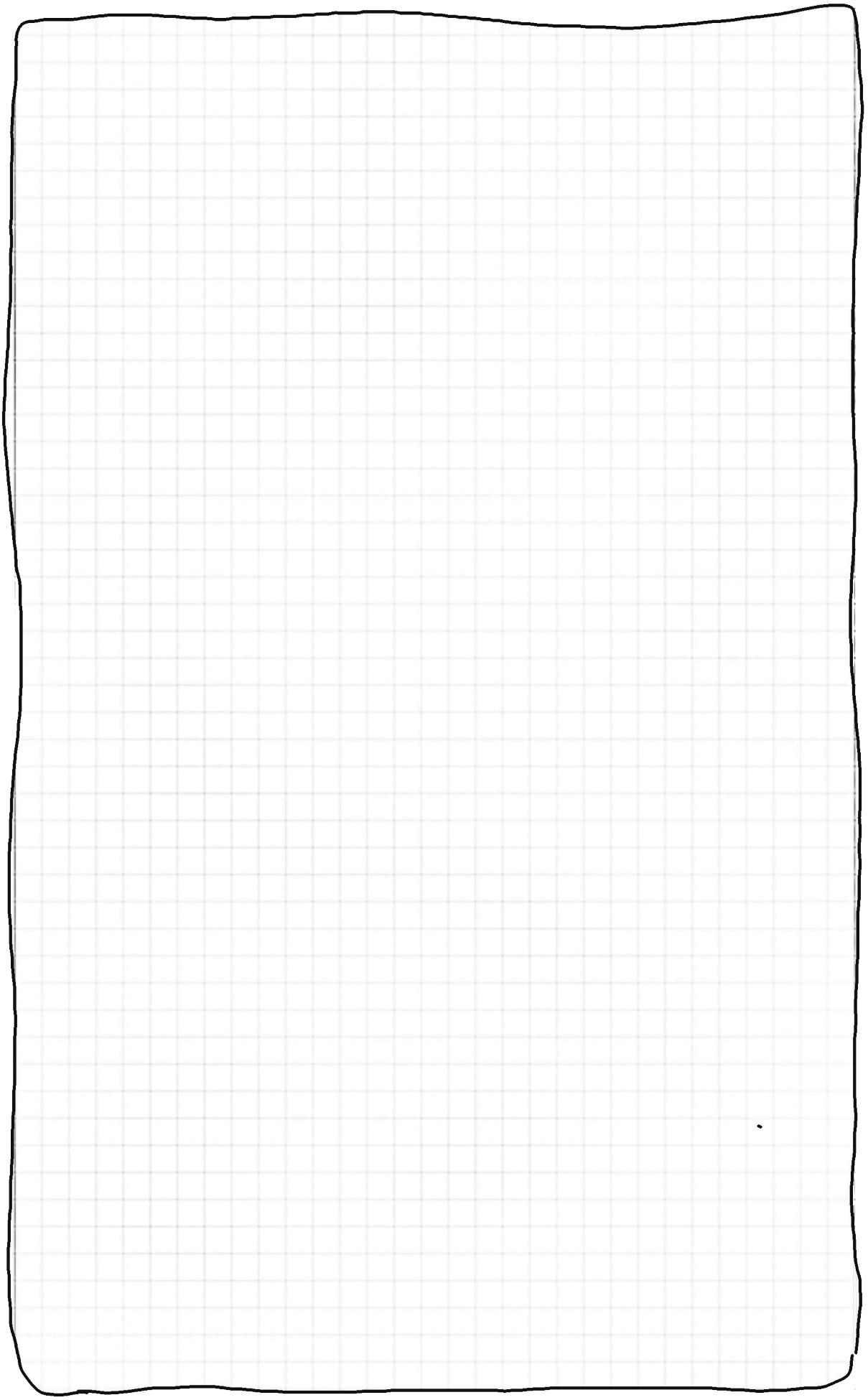
```
using KnetOnnx

# provide the ONNX  le's path:
# https://github.com/onnx/models
#
model = KnetModel("vgg16.onnx");

x = ones(224, 224, 3, 10)     # dummy input for prediction

model(x) # call KnetModel object with the model input,
         # the output is a 1000×10 Array{Float32,2}
```

ONNX Model Zoo

Open Neural Network Exchange (ONNX) is an open standard format for representing machine learning models. ONNX is supported by a community of partners who have implemented it in many frameworks and tools.

The ONNX Model Zoo is a collection of pre-trained, state-of-the-art models in the ONNX format contributed by community members like you. Accompanying each model are Jupyter notebooks for model training and running inference with the trained model. The notebooks are written in Python and include links to the training dataset as well as references to the original paper that describes the model architecture.

We have standardized on Git LFS (Large File Storage) to store ONNX model files. To download an ONNX model, navigate to the appropriate Github page and click the `Download` button on the top right.

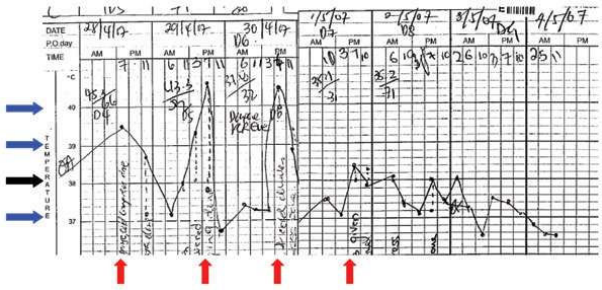


Kapitel 8:

RNN - Recurrent Neural Network

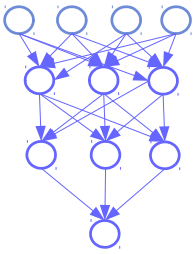
8.1 Regression mit neuronalen Netzen

Viele Zeitreihen haben skalare Größen



Ein Outputneuron genügt deshalb:

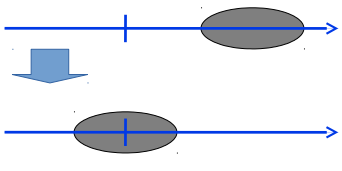
- Outputneurone:
- Hidden:
- Input:



Lineare Skalierung kann als min/max- oder z-Score-Skalierung durchgeführt werden)

Klimadaten:

```
1.5.2010  15.1
2.5.2010 13.3
3.5.2010 12.5
4.5.2010 17.4
5.5.2010 21.5
6.5.2010 18.3
7.5.2010 15.2
...
...
```

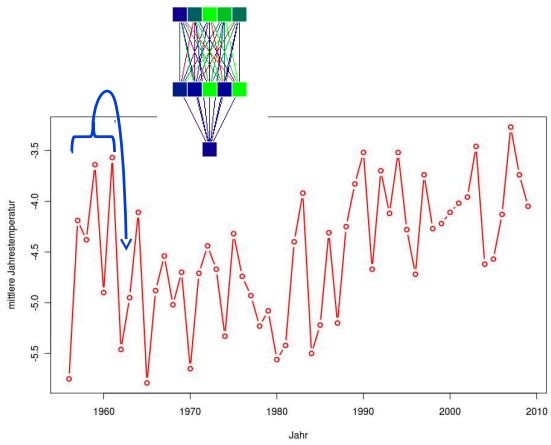


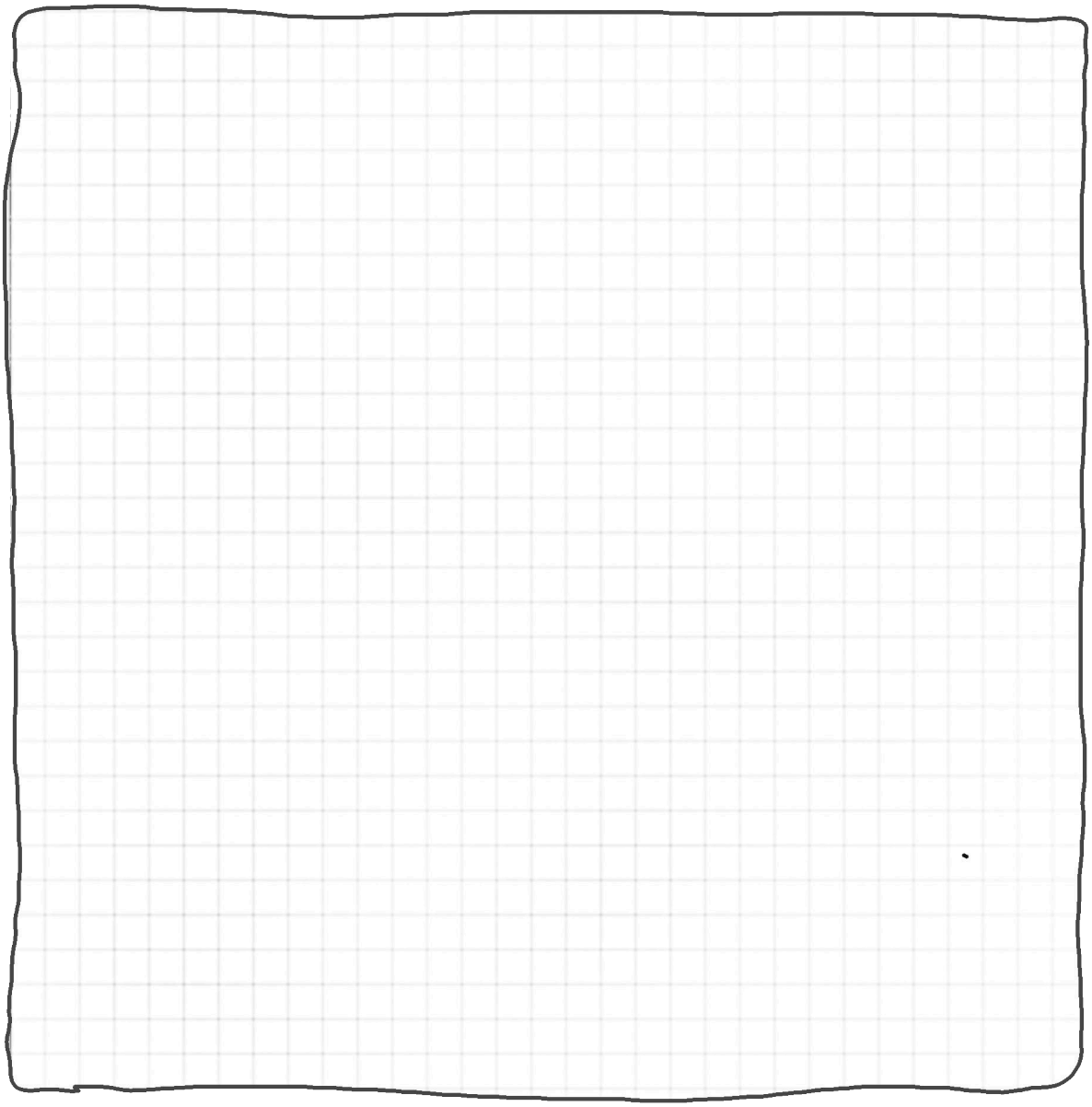
21.5 − 15.1 = 6.4 Skalierungsfaktor (*2)
(21.5 + 15.1) /2 = 18.3 Nullpunkt

Umrechnung Eingabe:
Input = (X-18.3)/3.2

8.2 Zeitreihen für Arme

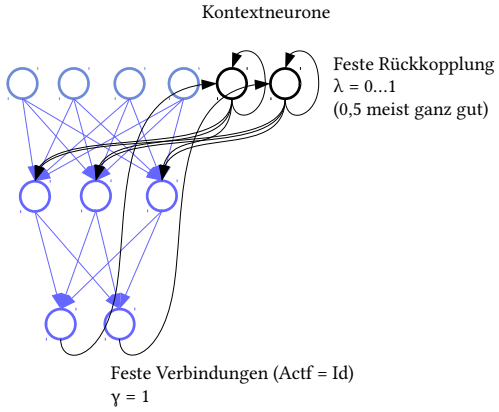
... sind mit einem MLP möglich:



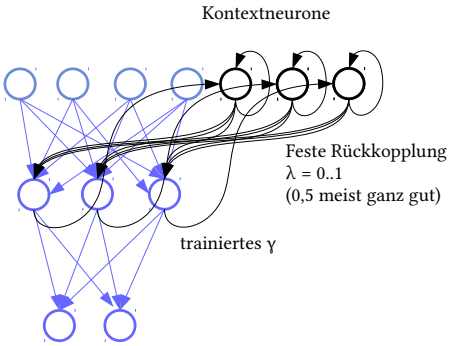


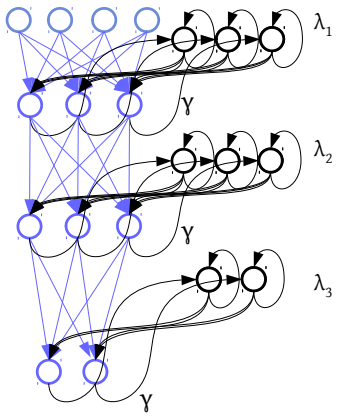
8.3 Kurzzeitgedächtnis mit Kontextneuronen

Jordan-Netz



Elman-Netz





Vorteile:

- Unterschiedliche λ pro Schicht.
- Je nach „Sinn“ werden die Kontextzellen mehr oder weniger genutzt!

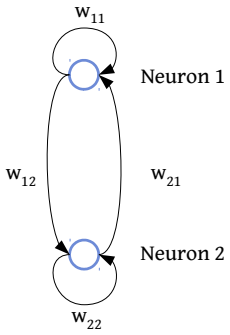
8.4 Backpropagation through time (BPTT)

Die Idee von Marvin Minsky

Idealisiertes rekurrentes Netz

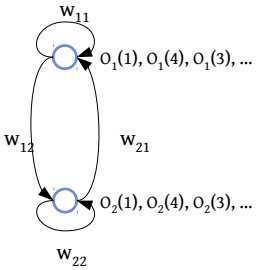
Training:

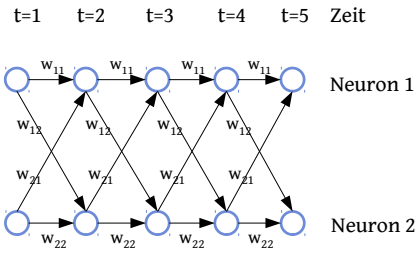
- Zeitreihe wird Schritt-für-Schritt angelegt (t=1, ..., n) und propagiert
- Dann mit Teaching Input verglichen
- Backprop muss durch alle Schritte (= Zeit) rückwärts gehen.



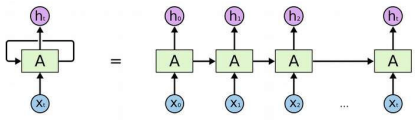
M. Minsky, S. Papert: Perceptrons, MIT Press, Cambridge, MA, 1-20, und 73, 1969.
J.A. Anderson, E. Rosenfeld (Eds.): Neurocomputing: Foundations of Research, Kap. 13, 161-170, MIT Press, 1988

unrolling des Netzes



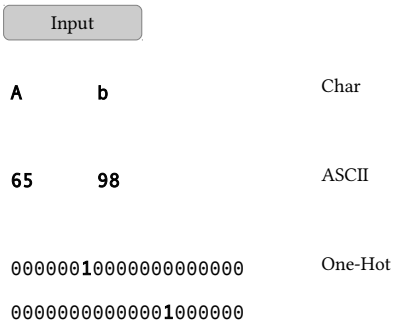


unrolling in den Bildchen von Chris Olah

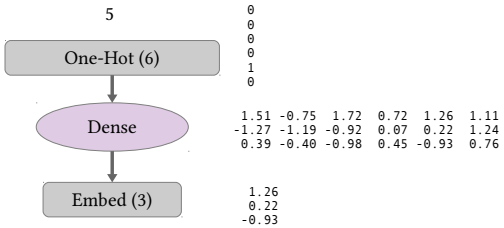


8.5 Embedding von Zeichen und Symbolen

Wie können Buchstaben kodiert werden?



One-Hot?



$$e = w * x$$

Multiplikation ist nicht notwendig!

$$e = w * x$$

```
   1     2     3     4     5     6
 1.51 -0.75  1.72  0.72  1.26  1.11       0
-1.27 -1.19 -0.92  0.07  0.22  1.24   *   0
 0.39 -0.40 -0.98  0.45 -0.93  0.76       0
                                          0
                                          1
                                          0
```

```
1.51*0 + -0.75*0 +  1.72*0 +  0.72*0 +  1.26*1 + 1.11*0 =  1.26
-1.27*0 + -1.19*0 + -0.92*0 +  0.07*0 +  0.22*1 + 1.24*0 =  0.22
 0.39*0 + -0.40*0 + -0.98*0 +  0.45*0 + -0.93*1 + 0.76*0 = -0.93
```

Vorschlag:

```
y = w[:,5]
```

Implementierung des Embedding:

```
using Knet: param, param0

struct Embed
    w
    actf
    Embed(i, e, actf=identity) = new(param(e,i), actf)
end

(l::Embed)(x) = l.actf.(l.w[:, x])
```

... nur noch achtgeben, wie x so immer aussieht!

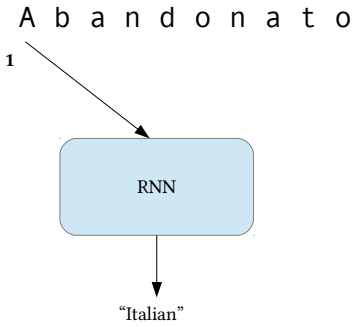
8.6 Beispiel: Erkennen der Sprache mit einem RNN

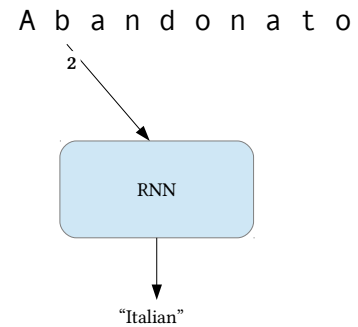
Datensatz

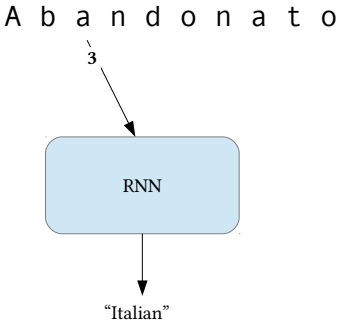
- Daten:

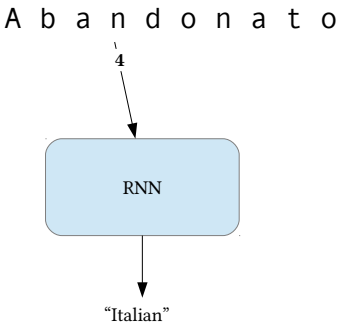
Arabic:	German:	Italian:
Abbing	Abbing	Abandonato
Abel	Abel	Abatangelo
Abeln	Abeln	Abatantuono
Abt	Abt	Abate
Achilles	Achilles	Abategiovanni
Achterberg	Achterberg	Abatescianni
Acker	Acker	Abbà
Ackermann	Ackermann	Abbadelli
Adam	Adam	Abbascia
Adenauer	Adenauer	Abbatangelo
Adler	Adler	Abbatantuono
Adlersflügel	Adlersflügel	Abbate
Aeschelman	Aeschelman	Abbatelli
Albert	Albert	Abbaticchio
Albrecht	Albrecht	Abbiati
Aleshire	Aleshire	Abbracciabene
Aleshite	Aleshite	Abbracciabeni
Althaus	Althaus	Abelli
Amsel	Amsel	Abelló
Andres	Andres	Abrami
...	...	...

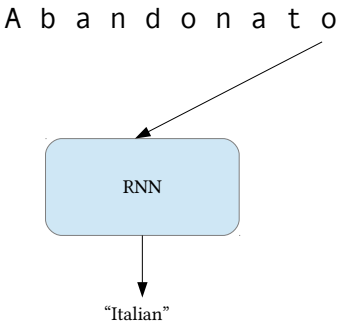
Das RNN soll die Buchstaben nacheinander verarbeiten:



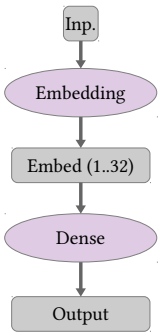



A b a n d o n a t o
3
RNN
“Italian”

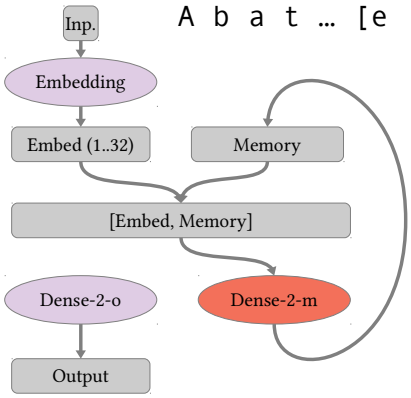

A b a n d o n a t o
4
RNN
“Italian”


A b a n d o n a t o
RNN
“Italian”

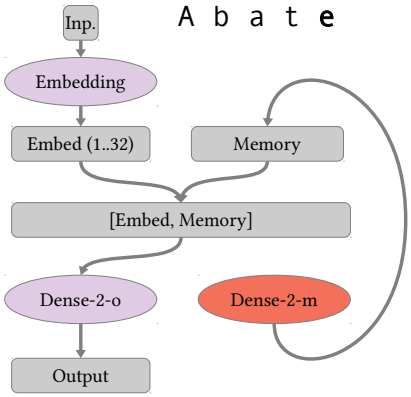
Das einfache Netz als MLP



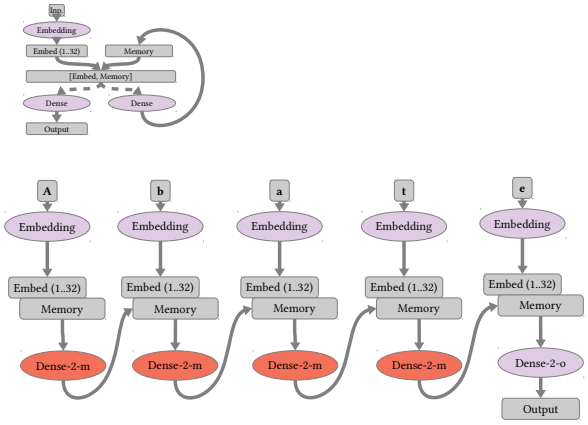
Besser als RNN: Ein Durchlauf pro Buchstabe



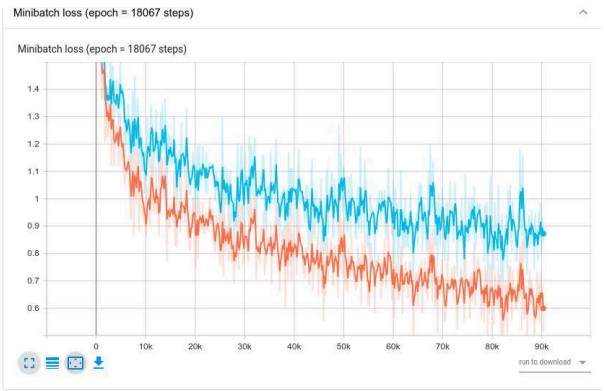
Erst am Ende wird der Output erzeugt:



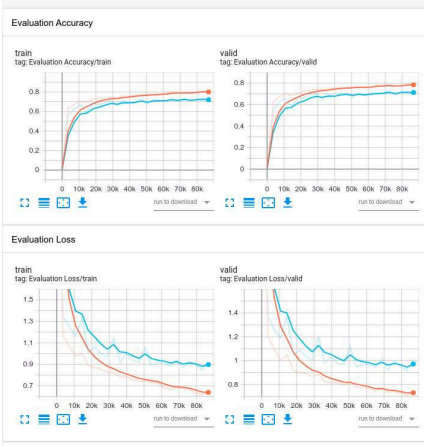
Unrolling und BPTT wir im normalen MLP



Im Video wird gezeigt, wie sich das sehr geradlinig umsetzen lässt.
Training mit Adam (rot) und SGD [lr=0.005, momentum=0,95], blau)



Das Training ist nach 100000 Steps noch nicht beendet:



8.7 Long Short-Term Memory, LSTM

LSTM, 1997

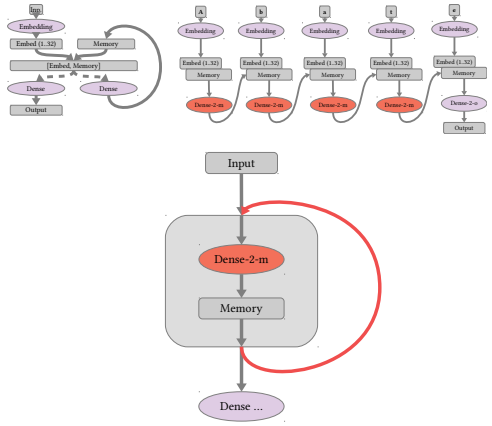
Die Lösung: Long Short-Term Memory Units (LSTMs)

Sepp Hochreiter and Juergen Schmidhuber

(You Again Shmidhoobuh, if you can say
Schwarzenegger & Schumacher & Schiffer, then you
can also say Schmidhuber)

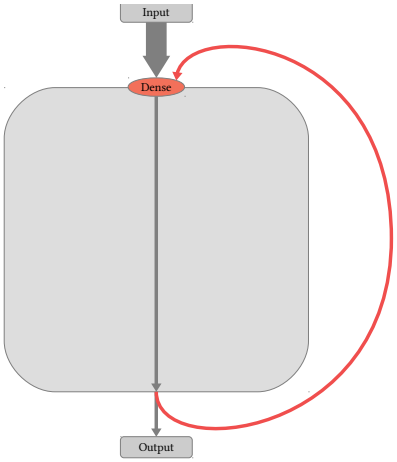
- Kann sich Aktivierung sehr lange merken
- Kann Aktivierung sehr schnell vergessen
- Addiert Veränderungen (statt Multiplikation)

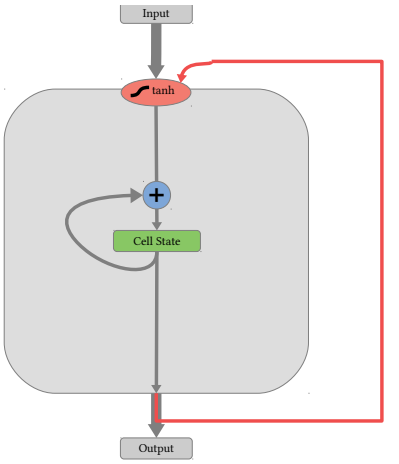
Recurrent Unit

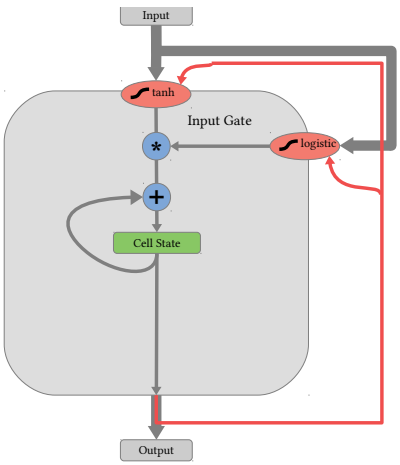


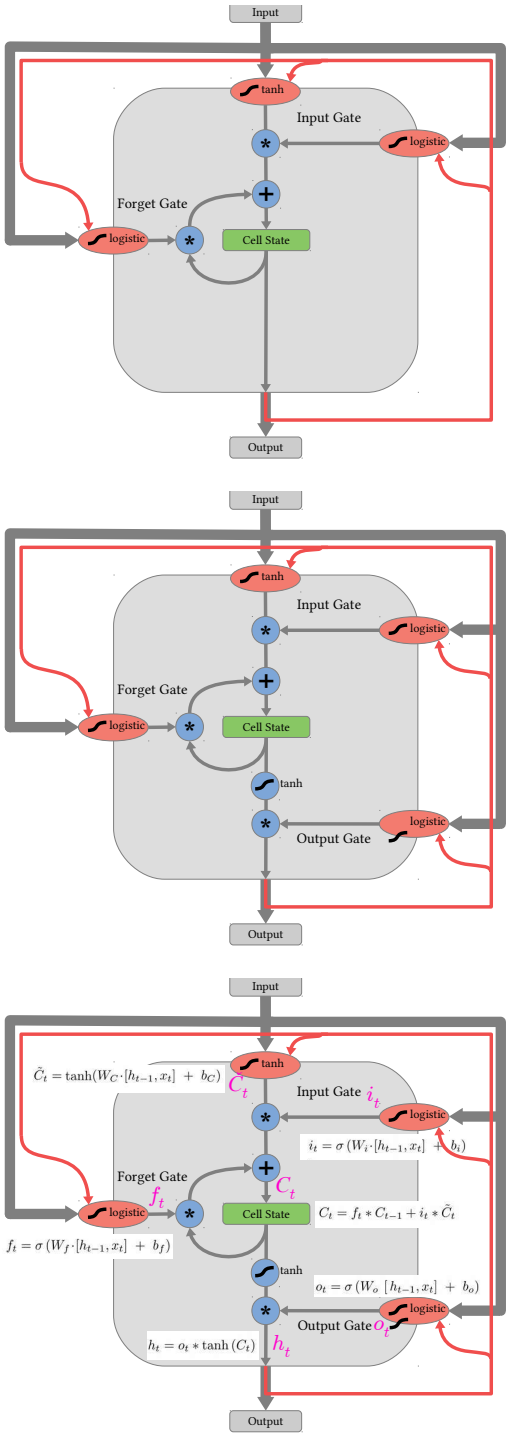
LSTM: Schritt für Schritt

LSTM: Schritt für Schritt





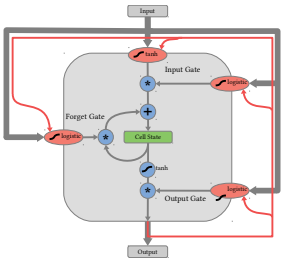



Input
tanh
Input Gate
logistic
Forget Gate
logistic
Cell State
Output
Input
tanh
Input Gate
logistic
Forget Gate
logistic
Cell State
tanh
logistic
Output Gate
Output
Input
$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$
tanh
C_t
Input Gate i_t
logistic
$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$
Forget Gate
f_t
C_t
logistic
Cell State
$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$
$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$
tanh
$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$
logistic
Output Gate o_t
$h_t = o_t * \tanh(C_t)$
h_t
Output

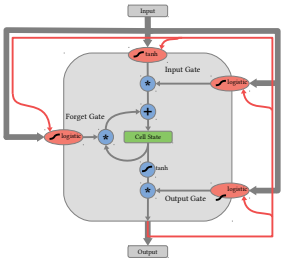
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$




```
f[t] = sigm(Wf * x[t] .+ Rf * h[t-1] .+ bWf .+ bRf)
i[t] = sigm(Wi * x[t] .+ Ri * h[t-1] .+ bWi .+ bRi)
C̃[t] = tanh(Wn * x[t] .+ Rn * h[t-1] .+ bWn .+ bRn)
C[t] = f[t] .* c[t-1] .+ i[t] .* C̃[t]
o[t] = sigm(Wo * x[t] .+ Ro * h[t-1] .+ bWo .+ bRo)
h[t] = o[t] .* tanh(C[t])
```



Vorteile:

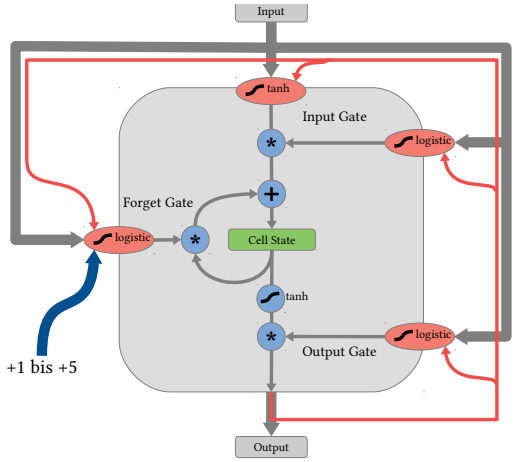
- unreasonable effective

Nachteile:

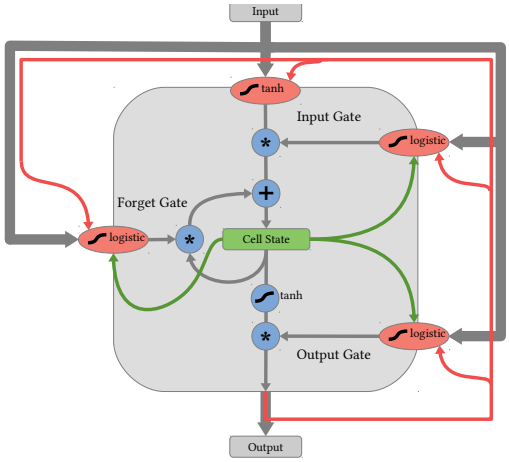
- viele Gewichte!
- tief
- viele kleine Tensoren ...
- ... die nacheinander berechnet werden müssen
- Minibatches?

Varianten des LSTM

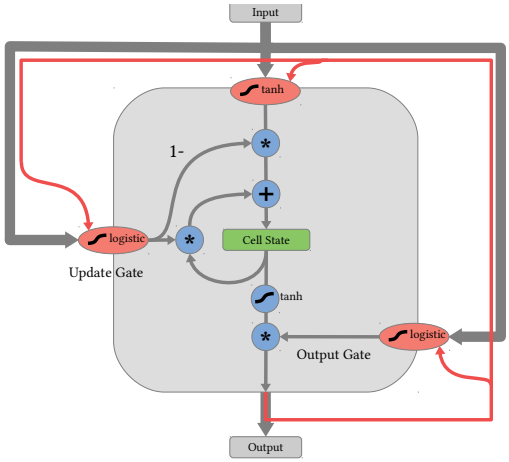
Erhöhung deds Forget-Bias



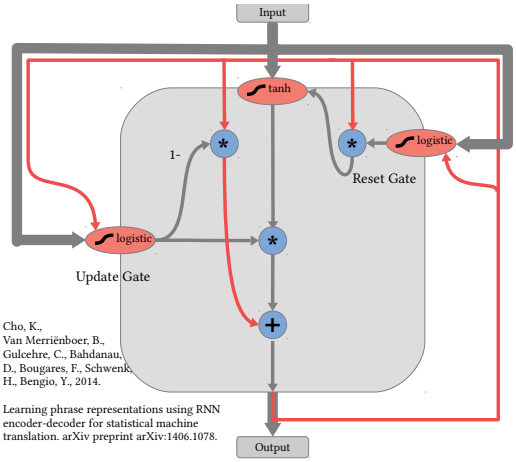
Peepholes



Update-Gate statt Input+Forget



GRU: Gated Recurrent Unit



Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., 2014.

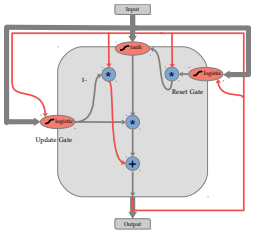
Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078.

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t] + b_r)$$

$$u_t = \sigma(W_u \cdot [h_{t-1}, x_t] + b_u)$$

$$\tilde{h}_t = \tanh(W_h \cdot [r_t * h_{t-1}, x_t] + b_h)$$

$$h_t = (1 - u_t) * h_{t-1} + u_t * \tilde{h}_t$$




```
u[t] = sigm(Wu * x[t] .+ Ru * h[t-1] .+ bWu .+ bRu)
r[t] = sigm(Wr * x[t] .+ Rr * h[t-1] .+ bWr .+ bRr)
ĥ[t] = tanh(Wh * x[t] .+ r[t] .* (Rh * h[t-1] .+ bRh) .+ bWh)
h[t] = (1 - u[t]) .* ĥ[t] .+ u[t] .* h[t-1]
```

Übersicht der Units:

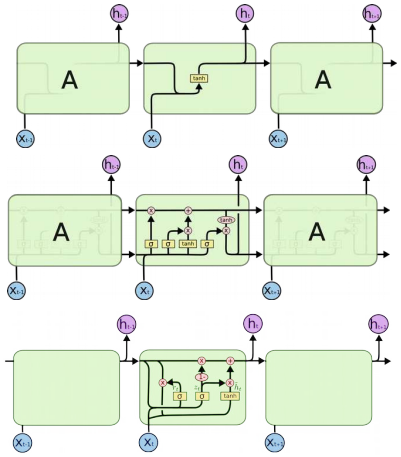
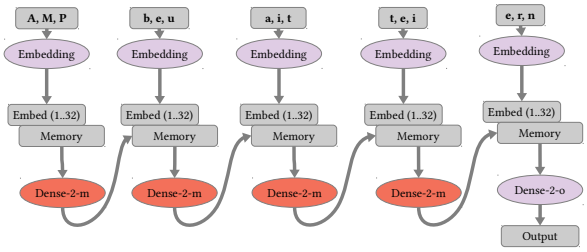


Abb.: Chris Olah's blog:
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

8.8 Architekturen für RNNs

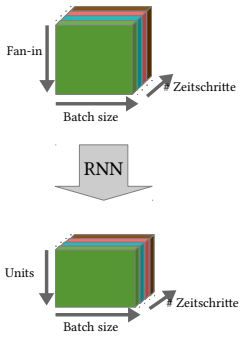
Minibatches in RNNs

- können nur pro Zeitschritt verwendet werden!
- gleich lang oder padding/truncate

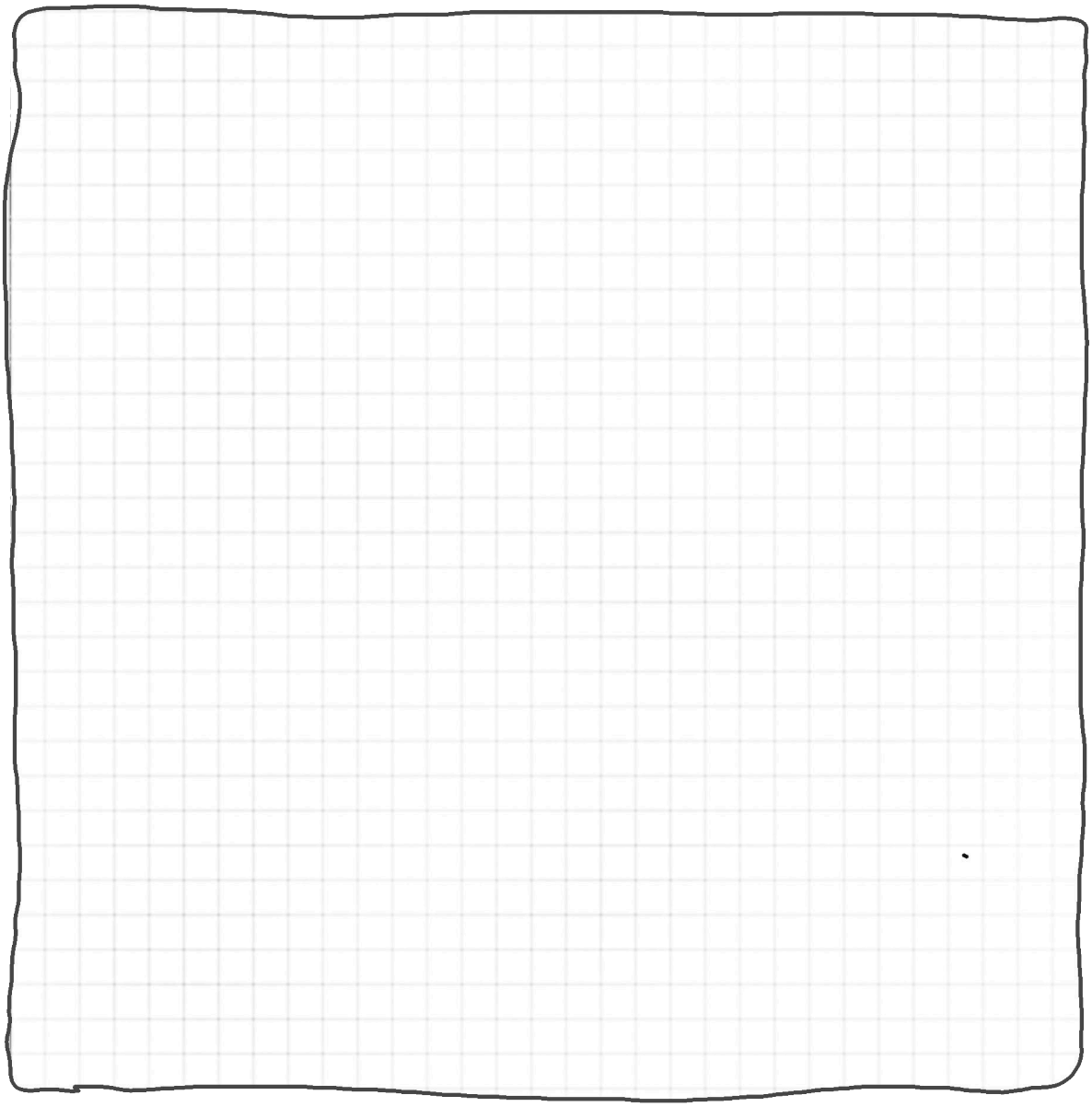


RNNs im Framework

- bieten Typen für LSTM, GRU oder andere recurrent Units.
- benötigen Inputtensoren: z.B. `Knet.RNN:`



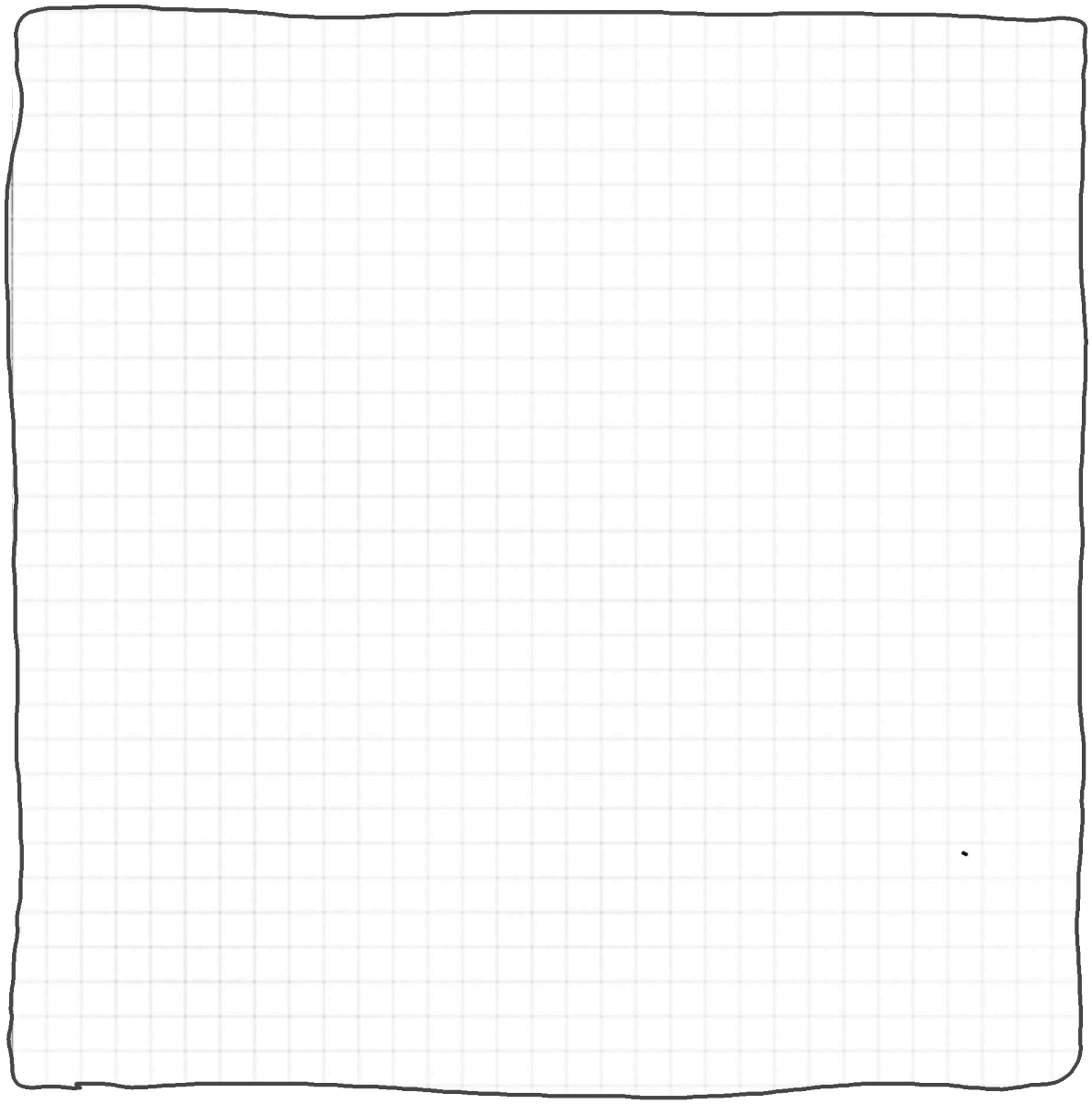
Sequence Tagger:



- Der Output aus jedem Zeitschritt wird benötigt!

```
struct RTagger
    n_inputs
    n_units
    unit_type
    rnn
    RNNTagLayer(n_inputs::Int, n_units::Int; u_type=:lstm) =
            new(n_inputs, n_units, u_type,
                Knet.RNN(n_inputs, n_units, rnnType=u_type))
end



function (rnn::Rtagger)(x)
    n_time_steps = size(x)[2]
    x = reshape(x, rnn.n_inputs, n_time_steps, :)
    x = permutedims(x, (1,3,2))     # [inputs, samples, time-steps]
    x = rnn.rnn(x)
    return permutedims(x, (1,3,2)) # [units, time-steps, samples]
end
```

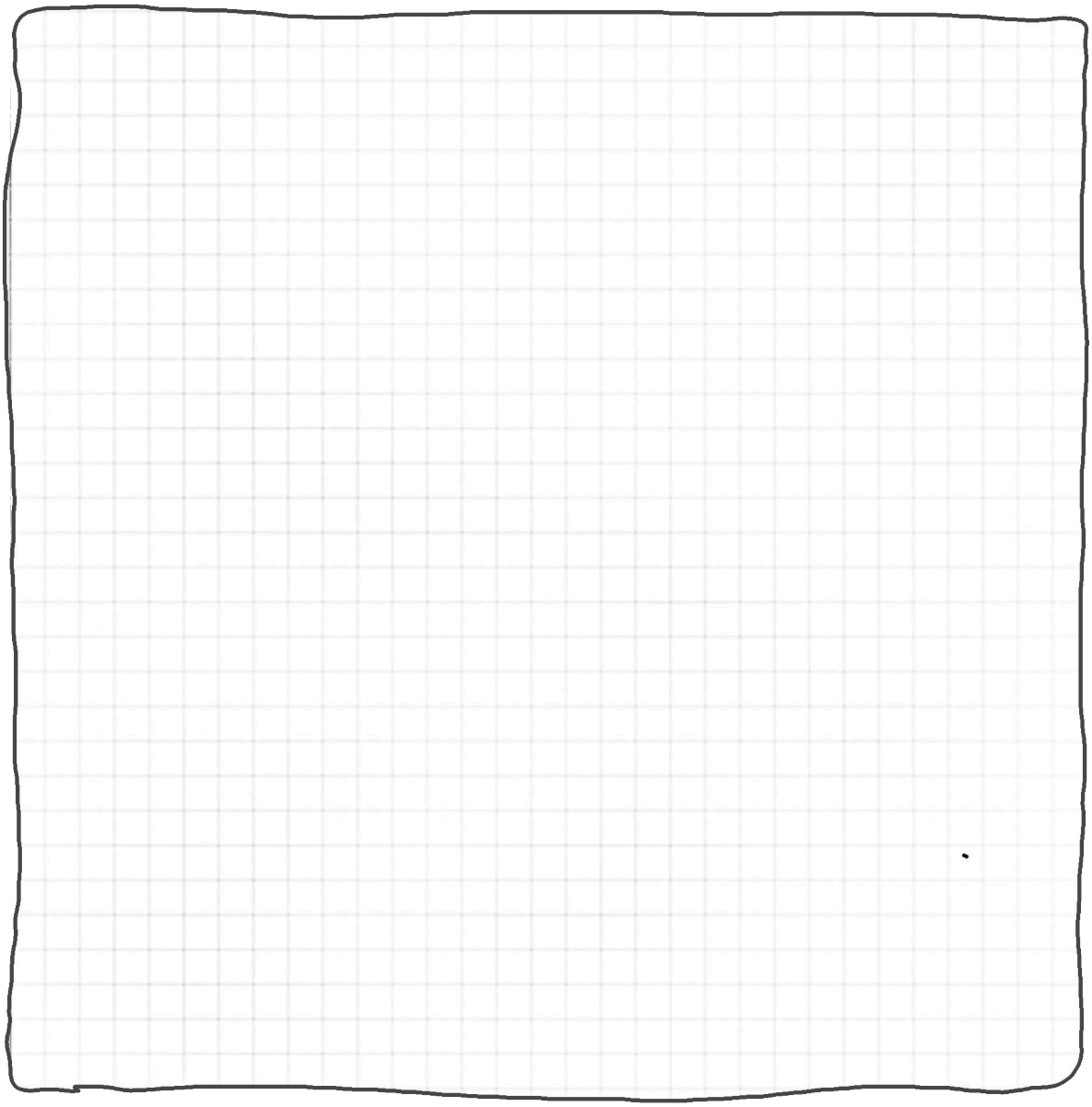
Sequence Classifier:



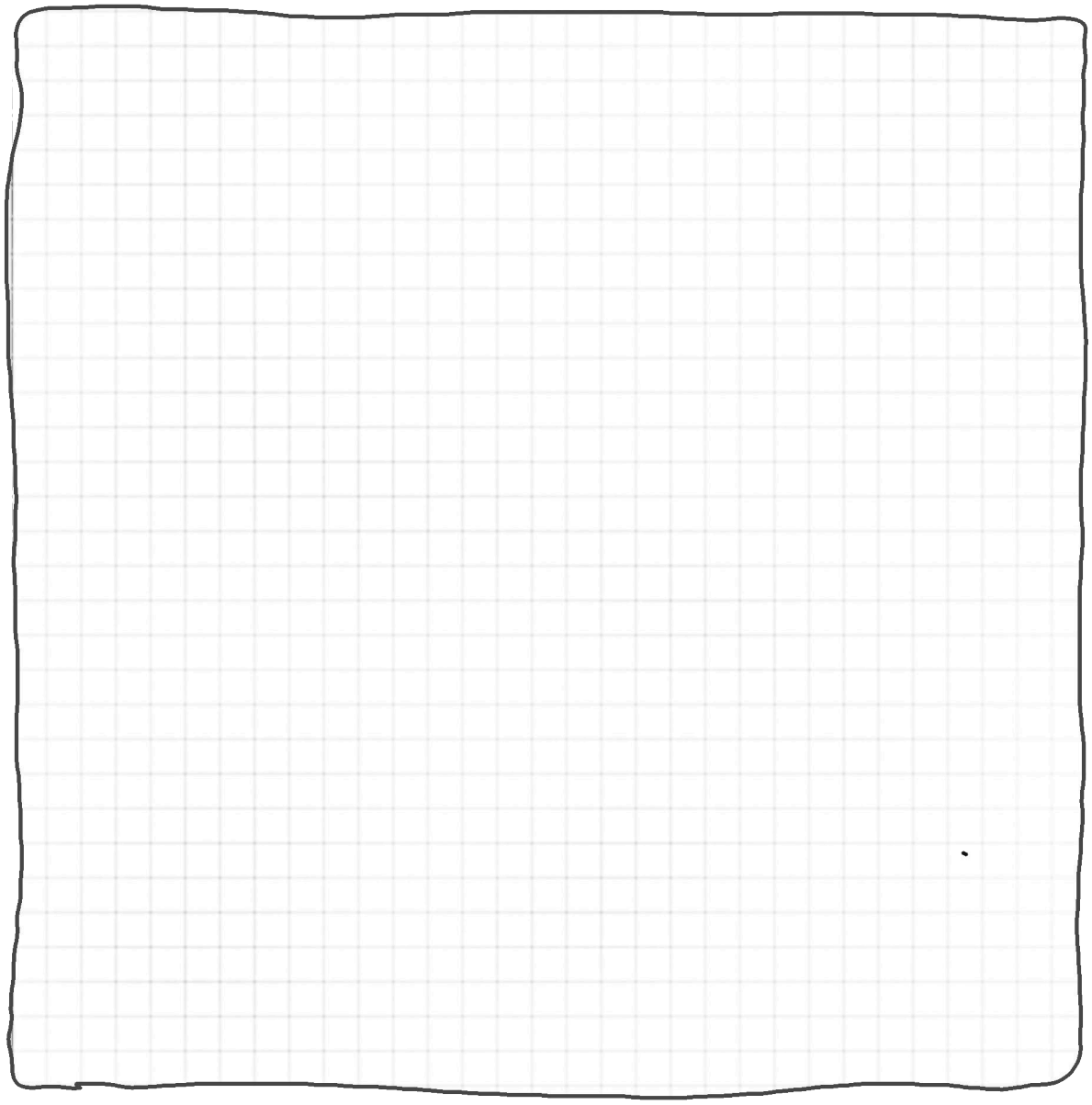
- Nur der letzte Output wird benötigt!

```
struct RSeqClassfr
    n_inputs
    n_units
    unit_type
    rnn
    RNNClassiLayer(n_inputs::Int, n_units::Int; u_type=:lstm) =
            new(n_inputs, n_units, u_type,
                Knet.RNN(n_inputs, n_units, rnnType=u_type))
end



function (rnn::RSeqClassfr)(x)
    n_time_steps = size(x)[2]
    x = reshape(x, rnn.n_inputs, n_time_steps, :)
    x = permutedims(x, (1,3,2))     # [inputs, time-steps, samples]
    x = rnn.rnn(x)
    return x[:,:,end]                    # [units, samples]
end
```

Attention:

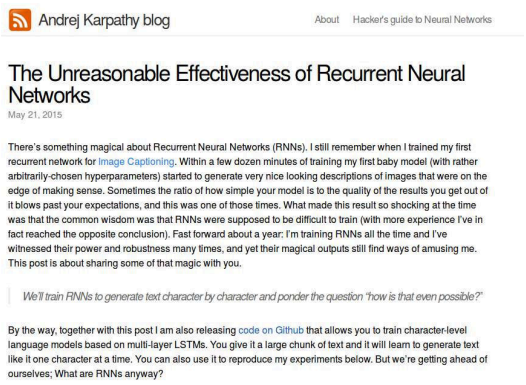


Bidirectional RNN:



8.9 Zusammenfassung

Es funktioniert



Andrej Karpathy blog About Hacker's guide to Neural Networks

The Unreasonable Effectiveness of Recurrent Neural Networks

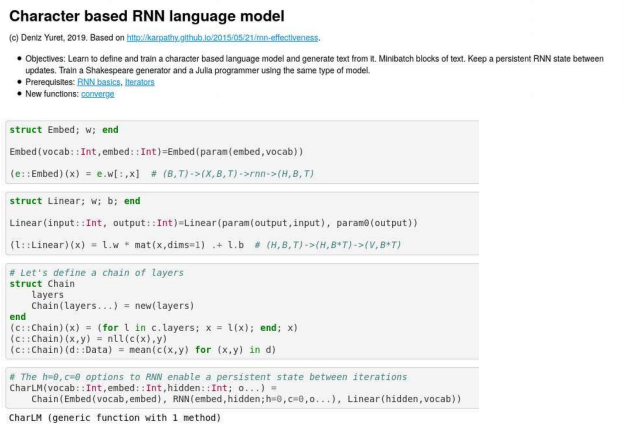
May 21, 2015

There's something magical about Recurrent Neural Networks (RNNs). I still remember when I trained my first recurrent network for Image Captioning. Within a few dozen minutes of training my first baby model (with rather arbitrarily-chosen hyperparameters) started to generate very nice looking descriptions of images that were on the edge of making sense. Sometimes the ratio of how simple your model is to the quality of the results you get out of it blows past your expectations, and this was one of those times. What made this result so shocking at the time was that the common wisdom was that RNNs were supposed to be difficult to train (with more experience I've in fact reached the opposite conclusion). Fast forward about a year: I'm training RNNs all the time and I've witnessed their power and robustness many times, and yet their magical outputs still find ways of amusing me. This post is about sharing some of that magic with you.

We'll train RNNs to generate text character by character and ponder the question "how is that even possible?"

By the way, together with this post I am also releasing code on Github that allows you to train character-level language models based on multi-layer LSTMs. You give it a large chunk of text and it will learn to generate text like it one character at a time. You can also use it to reproduce my experiments below. But we're getting ahead of ourselves; What are RNNs anyway?

Karpathys CharNN



Character based RNN language model

(c) Deniz Yuret, 2019. Based on http://karpathy.github.io/2015/05/21/rnn-effectiveness.

- Objectives: Learn to define and train a character based language model and generate text from it. Minibatch blocks of text. Keep a persistent RNN state between updates. Train a Shakespeare generator and a Julia programmer using the same type of model.
- Prerequisites: RNN basics, Iterators
- New functions: converge

```
struct Embed; w; end

Embed(vocab::Int,embed::Int)=Embed(param(embed,vocab))

(e::Embed)(x) = e.w[:,x]  # (B,T)->(X,B,T)->rnn->(H,B,T)
```

```
struct Linear; w; b; end

Linear(input::Int, output::Int)=Linear(param(output,input), param0(output))

(l::Linear)(x) = l.w * mat(x,dims=1) .+ l.b  # (H,B,T)->(H,B*T)->(V,B*T)
```

```
# Let's define a chain of layers
struct Chain
    layers
    Chain(layers...) = new(layers)
end
(c::Chain)(x) = (for l in c.layers; x = l(x); end; x)
(c::Chain)(x,y) = nll(c(x),y)
(c::Chain)(d::Data) = mean(c(x,y) for (x,y) in d)
```

```
# The h=0,c=0 options to RNN enable a persistent state between iterations
CharLM(vocab::Int,embed::Int,hidden::Int; o...) =
    Chain(Embed(vocab,embed), RNN(embed,hidden;h=0,c=0,o...), Linear(hidden,vocab))
```

```
CharLM (generic function with 1 method)
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

It looks like we can learn to spell English words. But how about if there is more structure and style in the data?

To examine this I downloaded all the works of Shakespeare and concatenated them into a single (4.4MB) file.

We can now afford to train a larger network, in this case lets try a 3-layer RNN with 512 hidden nodes on each layer.

After we train the network for a few hours we obtain samples such as:

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

The results above suggest that the model is actually quite good at learning complex syntactic structures. Impressed by these results, my labmate (Justin Johnson) and I decided to push even further into structured territories and got a hold of this book on algebraic stacks/geometry.

We downloaded the raw Latex source file (a 16MB file) and trained a multilayer LSTM. Amazingly, the resulting sampled Latex almost compiles. We had to step in and fix a few issues manually but then you get plausible looking math, it's quite astonishing:

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Lemma 0.1. *Assume (3) and (3) by the construction in the description.*

Suppose $X = \lim |X|$ *(by the formal open covering* X *and a single map* $\underline{Proj}_X(\mathcal{A}) = \mathrm{Spec}(B)$ *over* U *compatible with the complex*

$$Set(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \to \mathcal{C}_{Z/X}$ *is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition* ?? *(without element is when the closed subschemes are catenary. If* T *is surjective we may assume that* T *is connected with residue fields of* S*. Moreover there exists a closed subspace* $Z \subset X$ *of* X *where* U *in* X' *is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem*

(1) f *is locally of finite type. Since* $S = \mathrm{Spec}(R)$ *and* $Y = \mathrm{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X. But given a scheme U and a surjective étale morphism $U \to X$. Let $U \cap U = \coprod_{i=1,\ldots,n} U_i$ be the scheme X over S at the schemes $X_i \to X$ and $U = \lim_i X_i$. □

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{X,\ldots,0}$.

Lemma 0.2. *Let* X *be a locally Noetherian scheme over* S, $E = \mathcal{F}_{X/S}$. *Set* $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. *Since* $\mathcal{I}^n \subset \mathcal{I}^n$ *are nonzero over* $i_0 \leq \mathfrak{p}$ *is a subset of* $\mathcal{J}_{n,0} \circ \overline{A}_2$ *works.*

Lemma 0.3. *In Situation* ??. *Hence we may assume* $\mathfrak{q}' = 0$.

Proof. We will use the property we see that $\mathfrak{p}$ is the mext functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F-algebra where δ_{n+1} is a scheme over S. □

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

I wanted to push structured data to its limit, so for the final challenge I decided to use code.
In particular, I took all the source and header files found in the Linux repo on Github, concatenated all of them in a single giant file (474 MB of C code) (I was originally going to train only on the kernel but that by itself is only ~16MB).

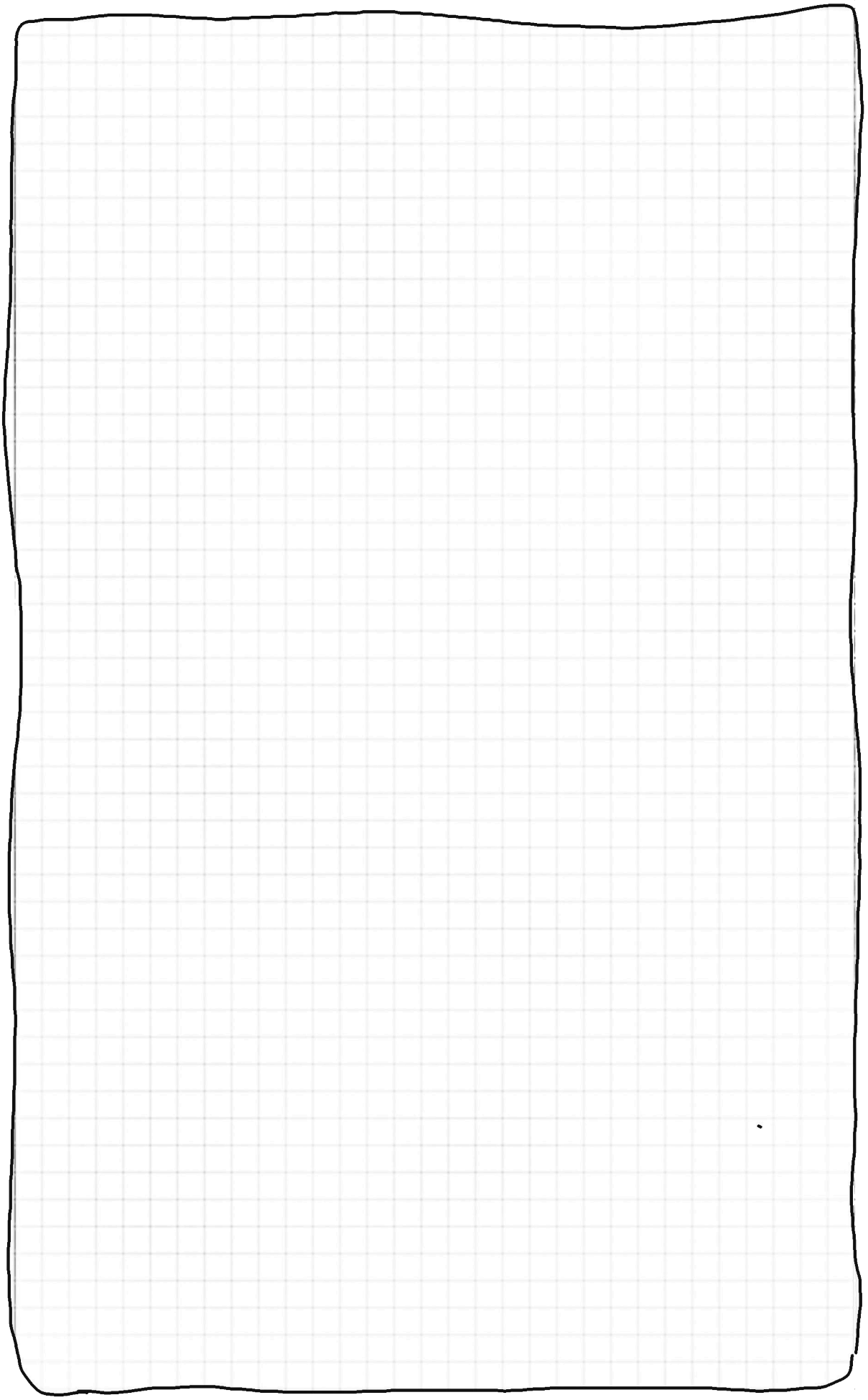
Then I trained several as-large-as-fits-on-my-GPU 3-layer LSTMs over a period of a few days. These models have about 10 million parameters, which is still on the lower end for RNN models. The results are superfun:

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
  int error;
  if (fd == MARN_EPT) {
    /*
     * The kernel blank will coeld it to userspace.
     */
    if (ss->segment < mem_total)
      unblock_graph_and_set_blocked();
    else
      ret = 1;
    goto bail;
  }
  segaddr = in_SB(in.addr);
  selector = seg / 16;
  setup_works = true;
  for (i = 0; i < blocks; i++) {
    seq = buf[i++];
    bpf = bd->bd.next + i * search;
    if (fd) {
      current = blocked;
    }
  }
  rw->name = "Getjbbregs";
  bprm_self_clearl(&iv->version);
  regs->new = blocks[(BPF_STATS << info->historidac)] | PFMR_CLOBATHINC_SECONDS << 12;
  return segtable;
}
```

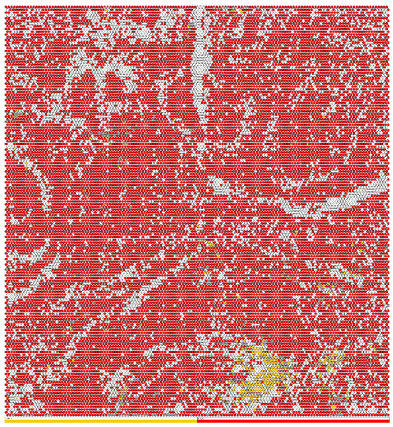
Fazit

- Unreasonable effective
- Unbelievable slow
- Hop or top!



Kapitel 9:

Kohonenkarten



Bildet einen Eingaberaum auf eine Fläche ab.

Lernt selbst, was wohin soll.

Nicht-linear!

Aliases:

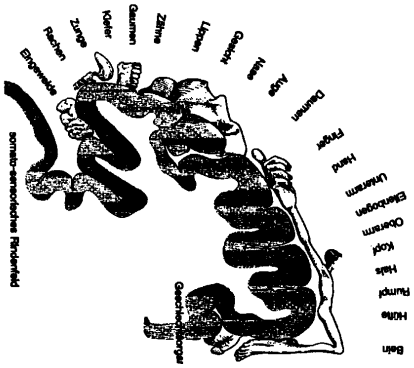
- Selbstorganisierende Karte
- self-organising map
- SOM
- Kohonen map
- Kohonen feature map

Von Teuvo Kohonen entwickelt (1982/84)

- Einschichtige neuronale Netze
- Neurone haben einen n-dimensionalen Gewichtsvektor
- Neurone sind über eine Abstandsfunktion verknüpft.

Somatosensorischer Hommunkulus

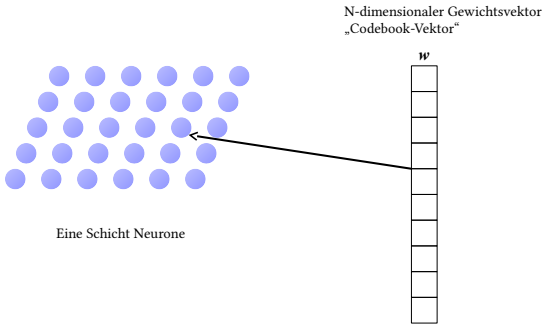
- Somato-sensorischer Kortex:
- Bildet unseren gesamten Körper auf eine Fläche ab
- Lernt selbst, was wohin soll
- Kein Teaching Input nötig

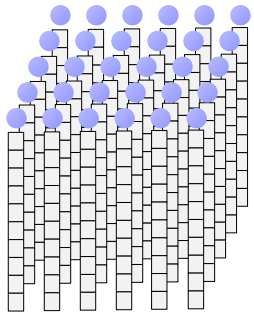


- Somato-sensorischer Kortex:
- Bildet unseren gesamten Körper auf eine Fläche ab
- Lernt selbst, was wohin soll
- Kein Teaching Input nötig

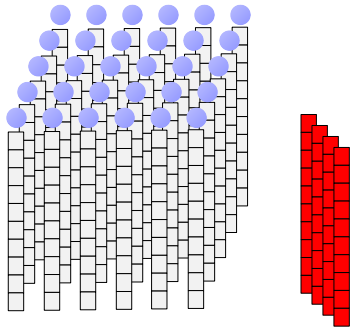
9.1 Training der SOM

Algorithmus – Schritt für Schritt



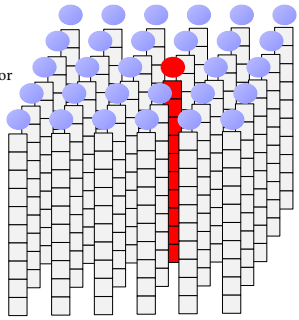


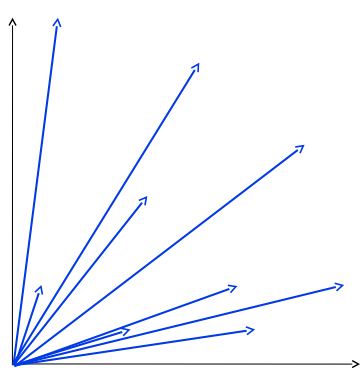
- Neurone werden nur durch ihre Codebookvektoren beschreiben
- ***Nicht*** durch ihren Aktivierungszustand
- ***Nicht*** durch ihre Verbindungen

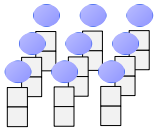


- Neurone werden nur durch ihre Codebookvektoren beschreiben
- ***Nicht*** durch ihren Aktivierungszustand
- ***Nicht*** durch ihre Verbindungen

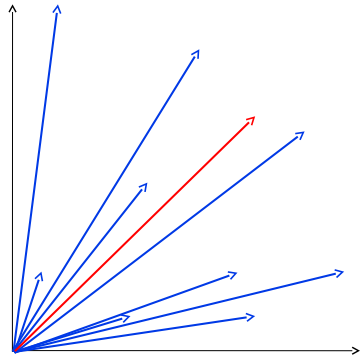
- Alle Codebookvektoren der Karte werden zufällig initialisiert
- Ersten Trainingsvektor wird mit den Vektoren der Karte vergleichen; wer passt am besten?
- Diesen Codebookvektor an den Trainingsvektor anpassen:
  "den Ähnlichsten noch ähnlicher machen"

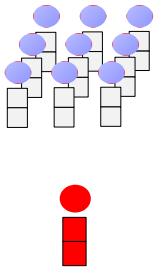




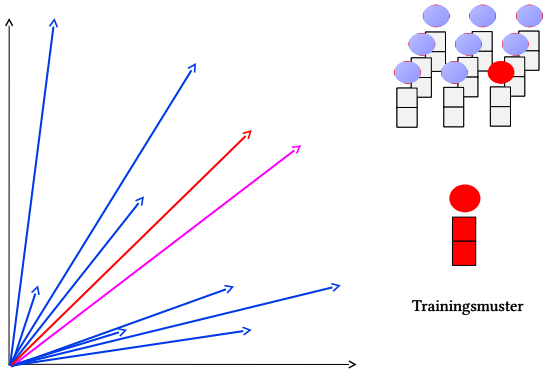


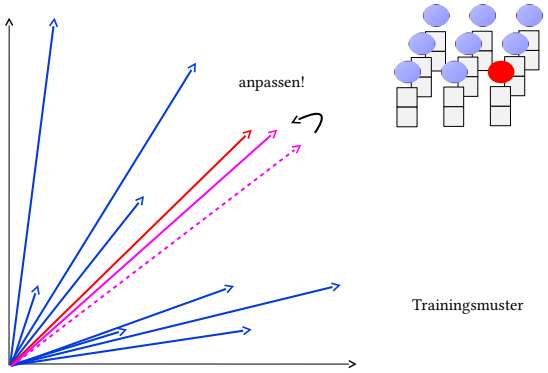
9 Neurone
9 Codebookvektoren





Trainingsmuster

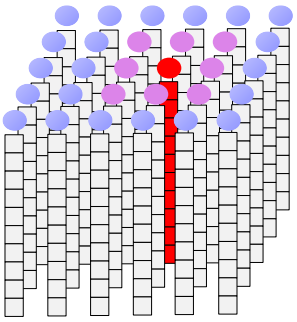


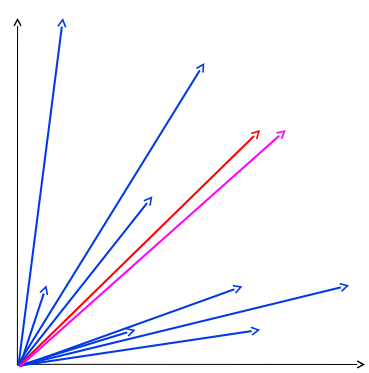


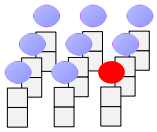
Umgebung

- Alle Codebookvektoren der Karte werden zufällig initialisiert
- Ersten Trainingsvektor wird mit den Vektoren der Karte vergleichen; wer passt am besten?

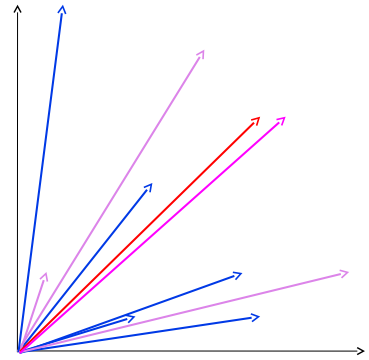
  "The winner takes it all"
- Diesen Codebookvektor an den Trainingsvektor anpassen:
  "den Ähnlichsten noch ähnlicher machen"
- Auch die Umgebung an den Trainingsverktor anpassen.

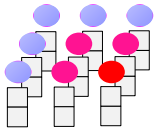


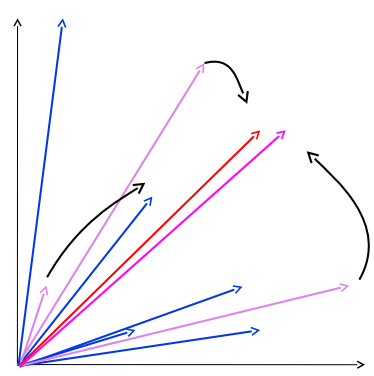


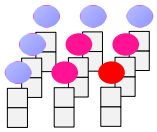


Trainingsmuster

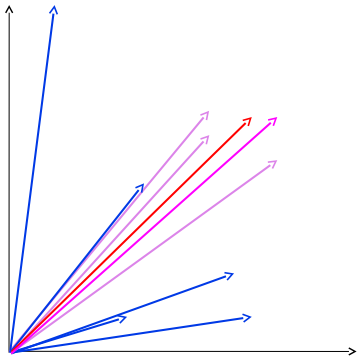


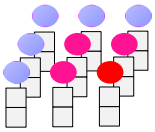






anpassen!



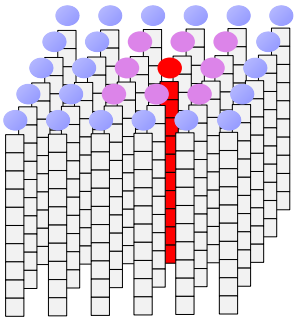


anpassen!

Algorithmus

- Alle Codebookvektoren der Karte werden zufällig initialisiert
- Ersten Trainingsvektor wird mit den Vektoren der Karte vergleichen; wer passt am besten?

  "The winner takes it all"
- Diesen Codebookvektor an den Trainingsvektor anpassen:
  "den Ähnlichsten noch ähnlicher machen"
- Auch die Umgebung an den Trainingsverktor anpassen
- Für alle Trainingsmuster wiederholen.



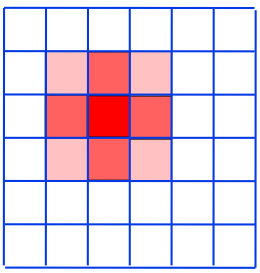
Abstandsfunktion

Auf rechteckigem Grid:

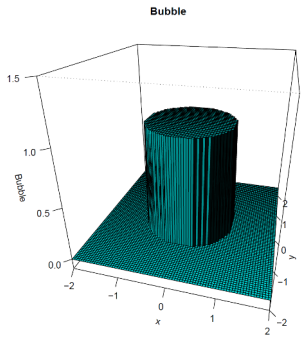
- von Neumann (4)
- **Moore (8)**

Generell:

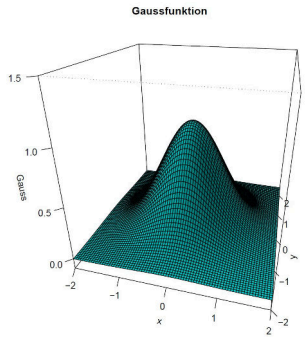
- **bubble**
- triagular
- **gaussian**
- mexican Heat



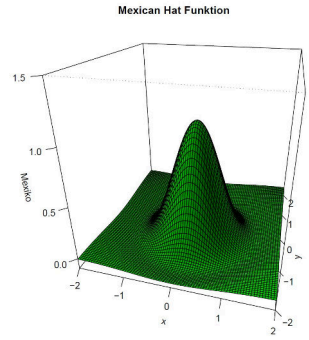
$$h = \begin{cases} 1 & \text{wenn } r < d \\ 0 & \text{sonst} \end{cases}$$



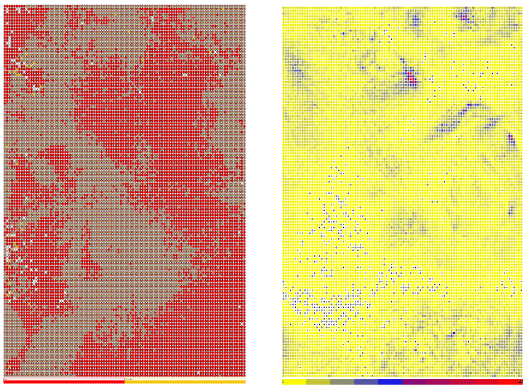
$$h = e^{-\left(\frac{r}{d}\right)^2}$$



$$h = \left(1 - \left(\frac{r}{d}\right)^2\right) \cdot e^{-\left(\frac{r}{d}\right)^2}$$



Beispiel



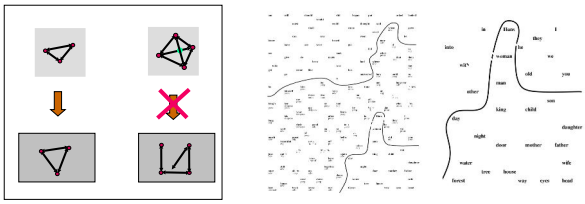
Unterschied SOM – PCA

MDS/PCA

- Versuch Punkte aus dem hochdimensionalen Raum in einen niedrig-dimensionalen Raum abzubilden!

SOM

- (nur) topologisch möglichst korrekt!



Plastiziät der SOM

Training mit den Koordinaten eines Quadrats:

1,0 0,0

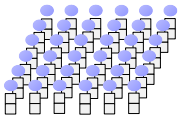
1,0 0,1

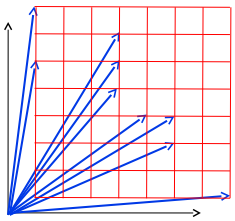
1,0 0,2

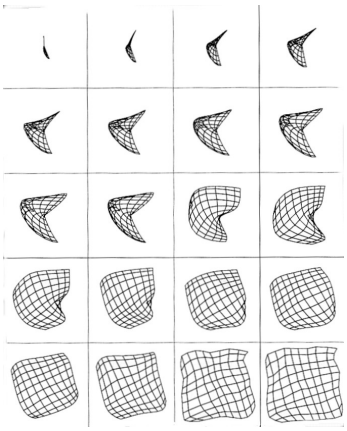
1,0 0,3

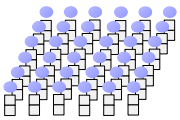
1,0 0,4

...

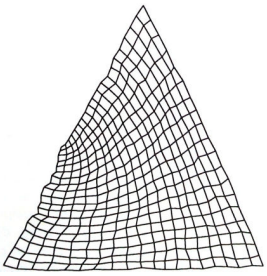


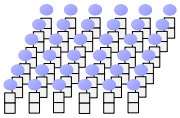






Quadratische Karte mit Dreieck als Eingaberaum





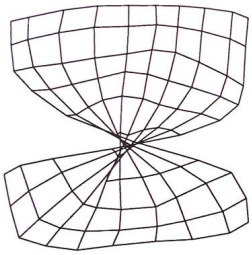
Auf der Seite `https://demogng.de/` von Bernd Fritzke lässt sich das sehr schön visualisieren!

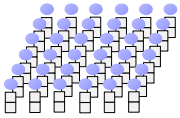
Die SOM passt sich den Daten an. Sie richtet sich entlang des größten Abstands aus, aber Defekte können auftreten:

Die SOM kann

- Knoten haben
- gespiegelt sein
- gedreht sein.

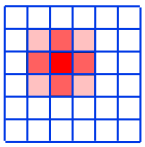
Karte mit topologischem Defekt





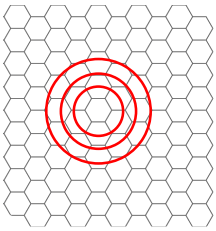
Rechtwinklig oder hexagonal:

2-dimensionale Kohonenkarten werden meist entweder im quadratischen oder im hexagonalen Gitter erstellt.



Quadratisch:

- Einfacher zu implementieren

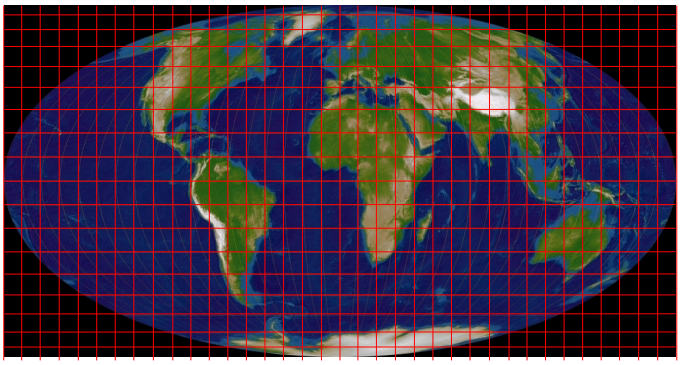


Hexagonal:

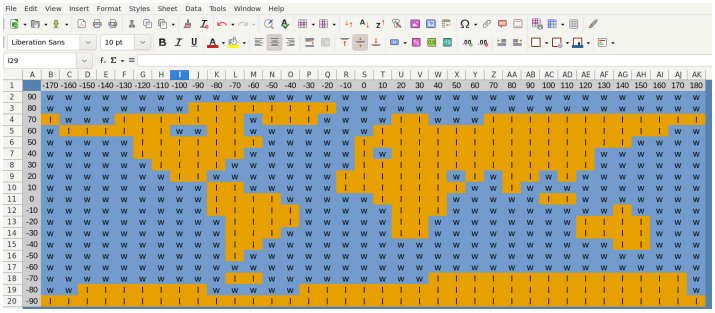
- Idealere Abstandsfunktion (regelmäßige „Sphären“ von Neuronen)
- Manchmal einfacher zu interpretieren
- Sieht schöner aus
- Sehr spitze Ecken.

9.2 Beispiele

Projektion des Globus



Die Erdkugel soll mit einer SOM auf eine flache Karte abgebildet werden!



468 Datenpunkte mit **Gradangaben**:

Ost/West Nord/Süd Land/Wasser

0 bis 180 0 bis 90 l oder w

0 bis -180 0 bis -90

Unsortiert!

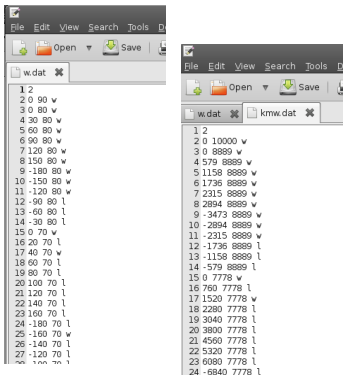
468 Datenpunkte mit **Kilometerangaben**:

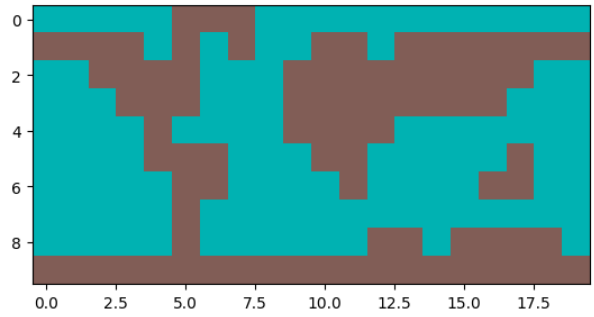
Ost/West Nord/Süd Land/Wasser

0 bis 20000 0 bis 10000 l oder w

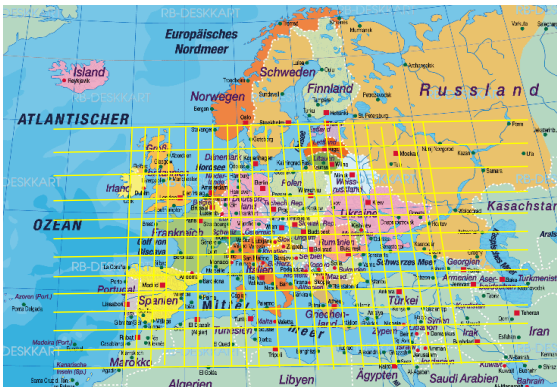
0 bis -20000 0 bis -10000

Unsortiert!





... mit Daten für Europa



468 Datenpunkte für die Welt mit Gradangaben:

Ost/West Nord/Süd Land/Wasser

0 bis 180 0 bis 90 l oder w

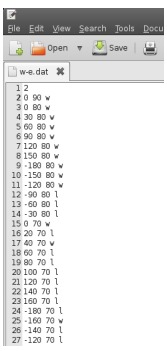
0 bis -180 0 bis -90

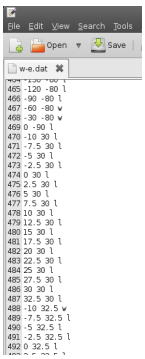
265 Datenpunkte für Europa mit Gradangaben:

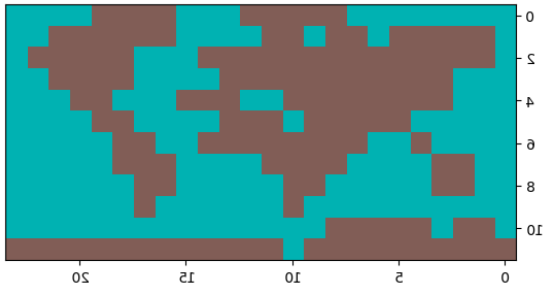
Ost/West Nord Land/Wasser

-10 bis 32,5 30 bis 65 l oder w

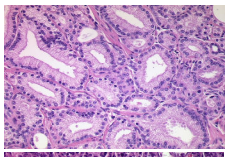
Unsortiert!





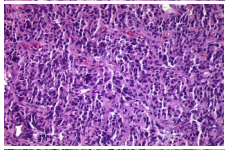


Beispiel: Diagnose Adenokarzinom der Postata



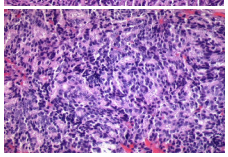
a) Gleason 3 samples:

	D_0	D_1	D_2	D_M
Minimum	1.379	1.411	1.282	1.300
Median	1.510	1.536	1.605	1.495
Maximum	1.598	1.673	1.745	1.593



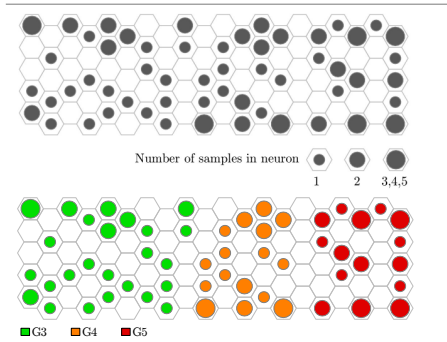
b) Gleason 4 samples:

	D_0	D_1	D_2	D_M
Minimum	1.594	1.590	1.595	1.546
Median	1.639	1.659	1.667	1.660
Maximum	1.715	1.771	1.834	1.743



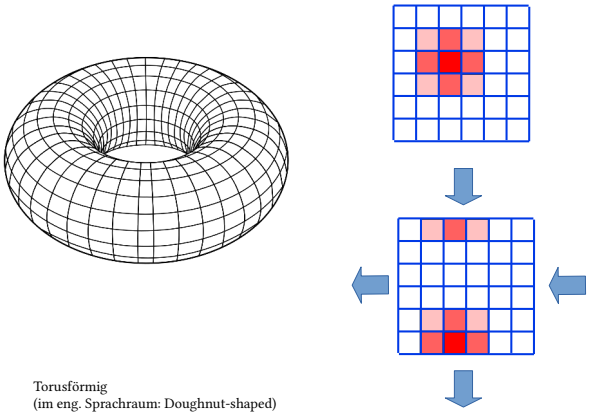
c) Gleason 5 samples:

	D_0	D_1	D_2	D_M
Minimum	1.712	1.699	1.744	1.687
Median	1.773	1.757	1.853	1.789
Maximum	1.857	1.836	1.901	1.925



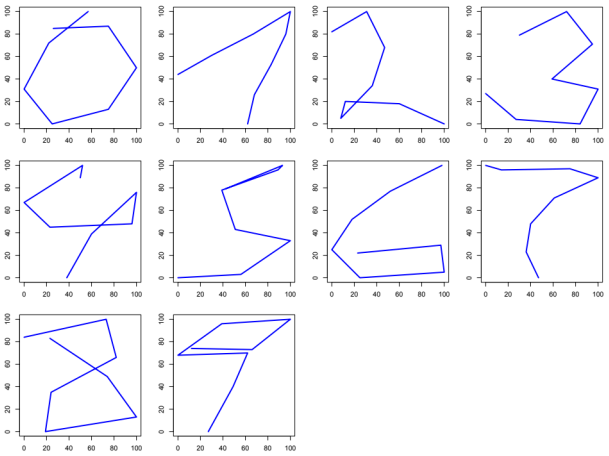
Przemyslaw Waliszewski, Dominik, A., Wagenlehner, F., Gattenloehner, S., Weidner, W., 2014. Complexity measures and classifiers in objective prostate cancer grading. 29th Annual Congress of the European Association of Urology Abstracts 13, e741.

Torus-förmige Karten sind endlos!

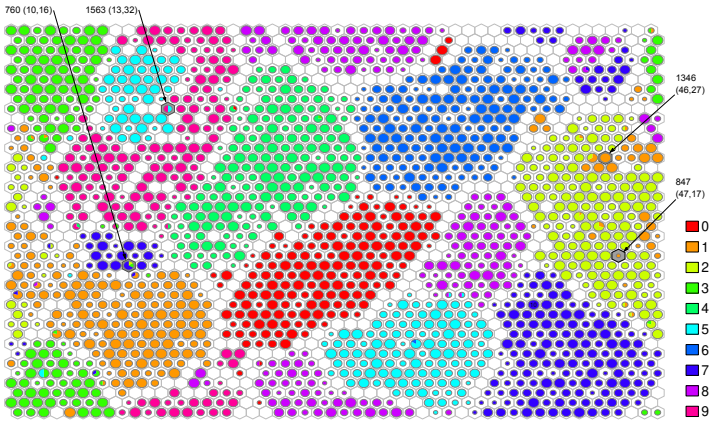


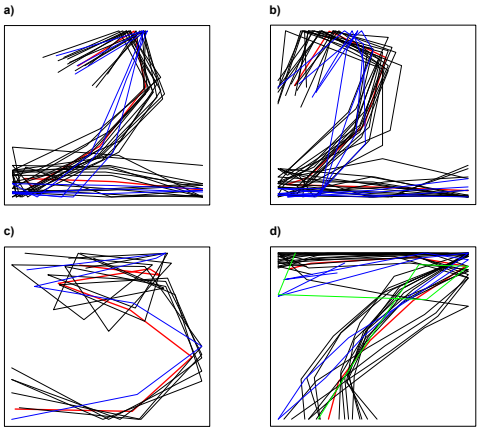
Torusförmig
(im eng. Sprachraum: Doughnut-shaped)

Beilspiel: Handwriting



.. mit dem R-Paket SOM_NN erstellt!



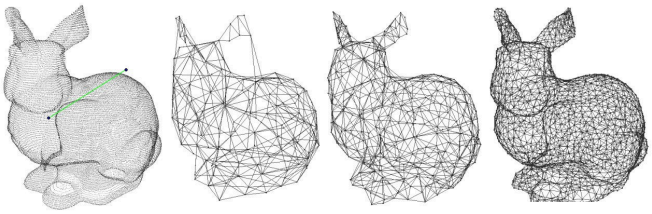


9.3 Zusammenfassung

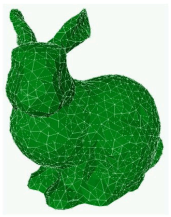
Tipps

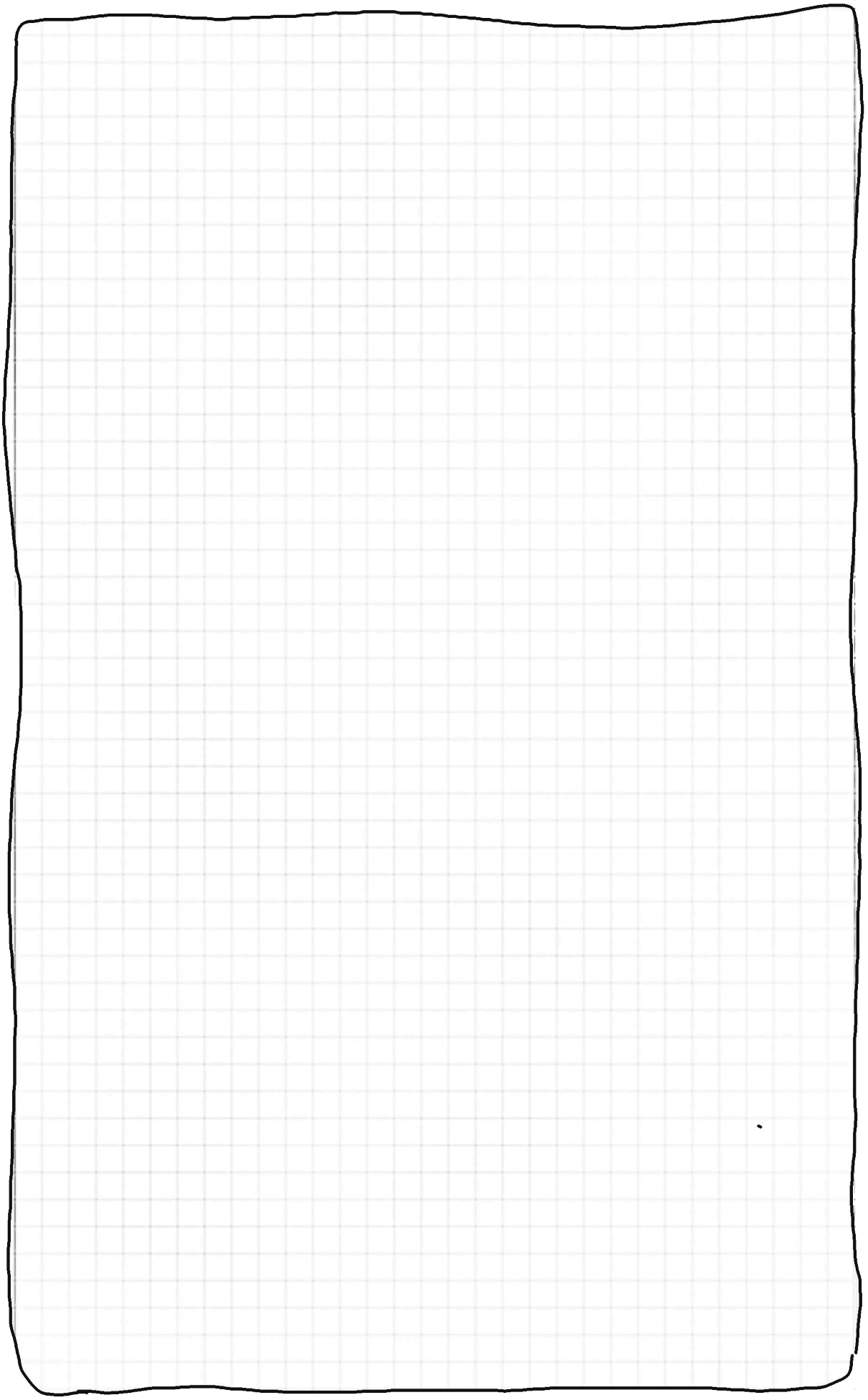
- **Dimension der Eingabedaten:**
  Beliebig. 1 – n*1000 sind möglich, mehrere 100 sind durchaus praktikabel und führen noch zu guten Karten. Die SOM passt sich dem Problem an und selektiert wichtige von unwichtigen Komponenten selbständig.
- **Dimension der Karte:**
  2-dimensionale Karten sind am besten visualisierbar. Höhere Dimension sollte nur verwendet werden, wenn das Problem dies erfordert.
- **Gitter:**
  Kohonen selbst empfiehlt hexagonale Karten wegen der klaren Abstandsfunktion. Und: ***die sehen einfach besser aus***. Quandratische Gitter sind aber einfacher zu implementieren.
- **Form der Karte:**
  Nicht exakt quadratisch, da die Karte sonst keine Vorzugsachse hat. Das Training dauert dann länger. Doughnut-förmige Karten (ohne Rand) sind nur manchmal besser.
- **Zahl der Lernschritte:**
  Ist immer sehr groß: 10.000 – 1.000.000. Es spielt meist keine Rolle in welcher Reihenfolge die Muster präsentiert werden.
- **Verstärkung seltener Fälle:**
  Da sich die Karte dem Datensatz anpasst, gehen seltene beim Training Fälle verloren. Sind diese wichtig (z.B.: 1 fehlerhaftes Produkt aus 100000 soll erkannt werden), dann müssen diese Muster entsprechend häufiger präsentiert werden.

Inzwischen gibt es viele Modifikationen, die wir auch noch besprechen werden!



- Learning Vecor Quantisation
- Neurales Gas
- Growing Neural Gas.





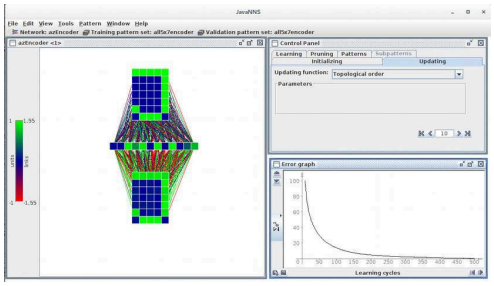
Kapitel 10:

Autoencoder

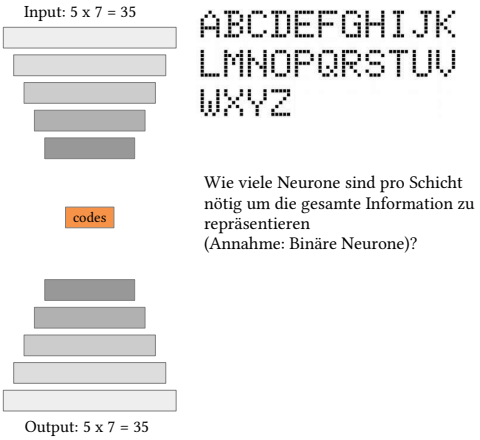
10.1 Vanilla Autoencoder

Ein Encoder und Decoder

Vanilla Autoencoder für die 5x7-Buchstaben funktioniert sofort

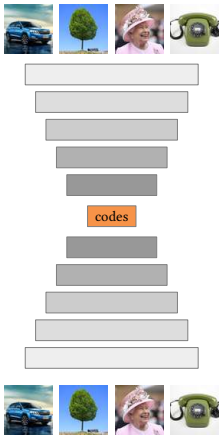


Wieviele Codes sind notwendig für 26 Buchstaben?



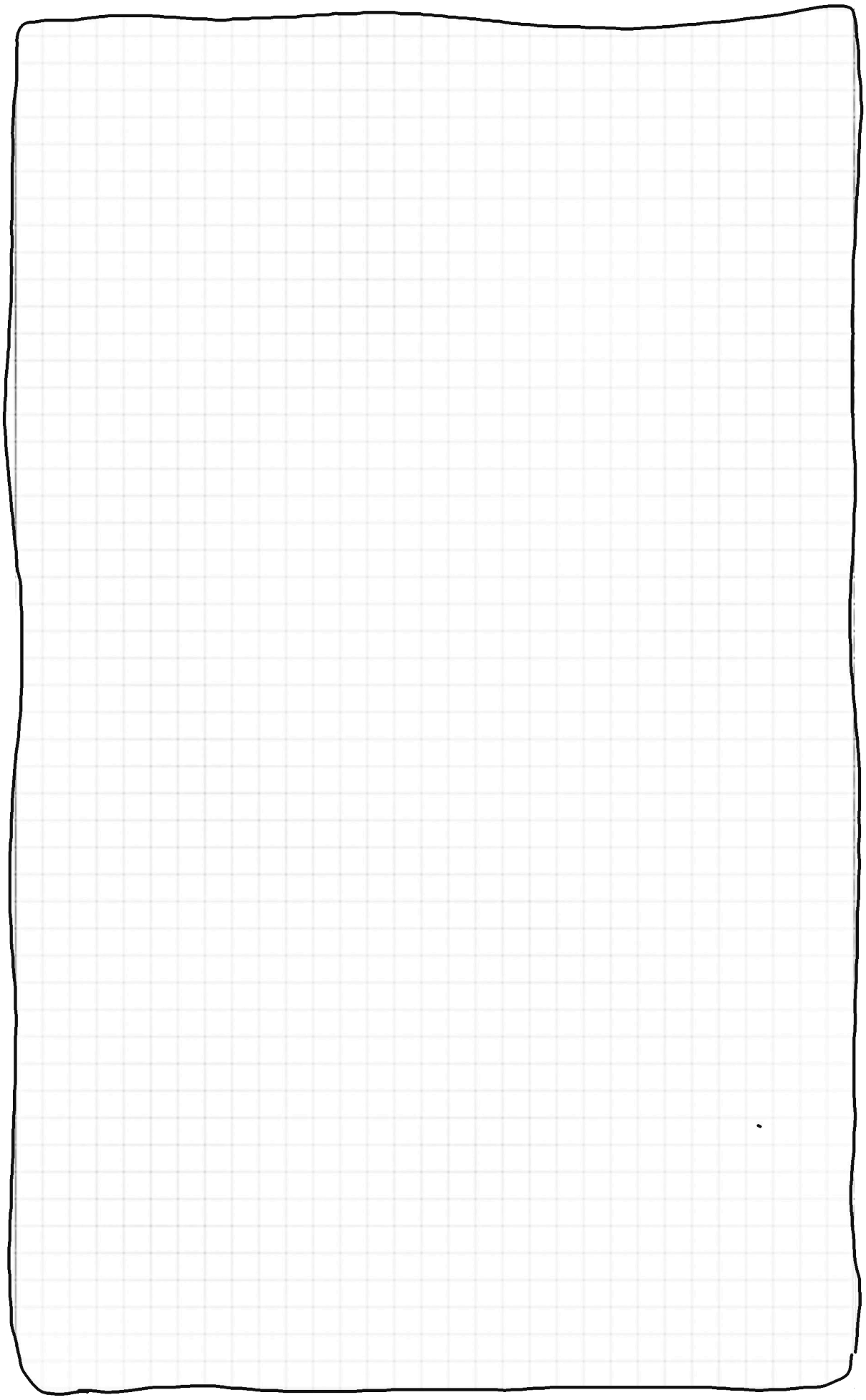
Wie viele Neurone sind pro Schicht nötig um die gesamte Information zu repräsentieren
(Annahme: Binäre Neurone)?

Wieviele Codes sind notwendig für 4 Bilder?



Tutorial: so wird's umgesetzt

Code:



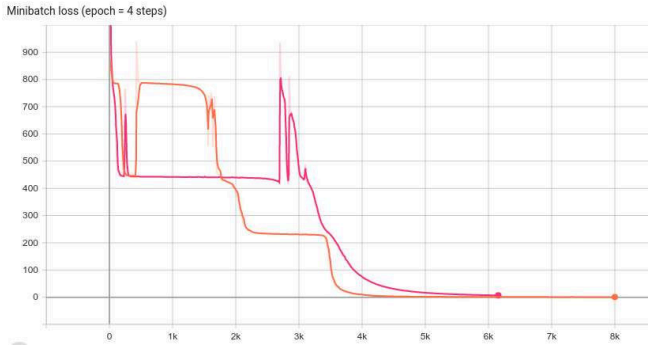
... wenige Zeilen Code ...

Model:

In [52]:
```
leaky(x) = sigm(x) + Float32(0.001)*x
actf = leaky
encode = Chain(
             Dense(12288,1024, actf=actf),
             Dense(1024, 256, actf=actf),
             Dense(256, 4, actf=actf))

decode = Chain(
            Dense(4, 256, actf=actf),
            Dense(256, 1024, actf=actf),
            Dense(1024, 12288, actf=actf))


mdl = Regressor(encode, decode)
```

Out[52]:
```
Regressor((Chain((Dense(P(KnetArray{Float32,2}(1024,12288)), P(KnetArr
ay{Float32,1}(1024)), leaky), Dense(P(KnetArray{Float32,2}(256,1024)),
P(KnetArray{Float32,1}(256)), leaky), Dense(P(KnetArray{Float32,2}(4,2
56)), P(KnetArray{Float32,1}(4)), leaky))), Chain((Dense(P(KnetArray{F
loat32,2}(256,4)), P(KnetArray{Float32,1}(256)), leaky), Dense(P(KnetA
rray{Float32,2}(1024,256)), P(KnetArray{Float32,1}(1024)), leaky), Den
se(P(KnetArray{Float32,2}(12288,1024)), P(KnetArray{Float32,1}(1228
8)), leaky)))))
```

... und wilde Lernkurven ...



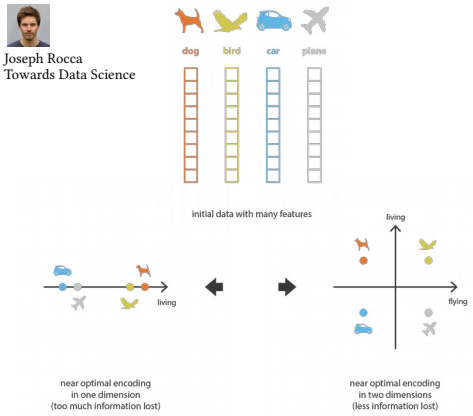
10.2 Variational Autoencoder

AEs können wide, tight, super tight und deep sein!

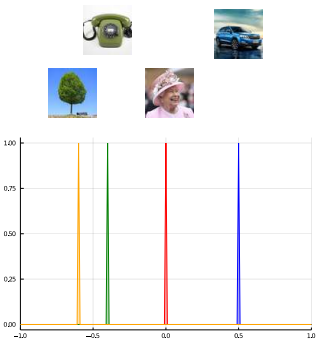
Übersicht

- **Regularisierung** → **VAE**
- Tiefe → DBN
- MLP → CNN-AE
- Informationsverlust im Bottleneck → uNet
- MLP → Seq2Seq

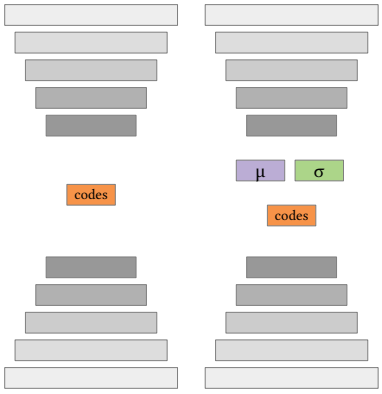
Vergleich von AE und PCA zur Dimensionsreduktion: ist nicht vergleichbar!



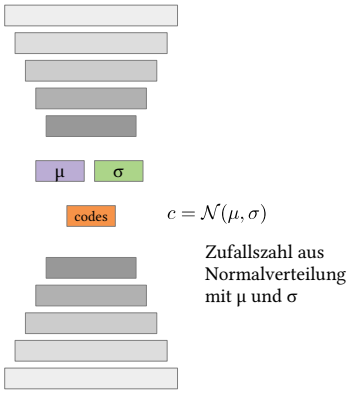
Beim MLP gibt es wenig Regularisierung; dafür viel Overfitting



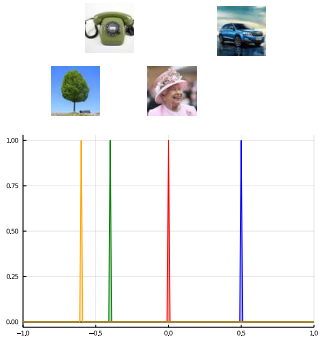
Aber Regularisierung ist im Bottleneck möglich.

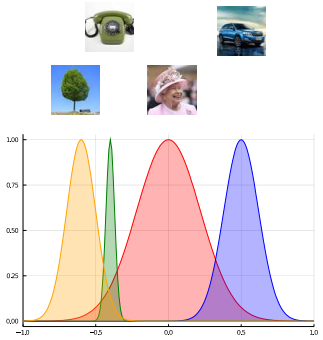


Der Code wird durch eine Zufallszahlen (normalverteilt) ersetzt

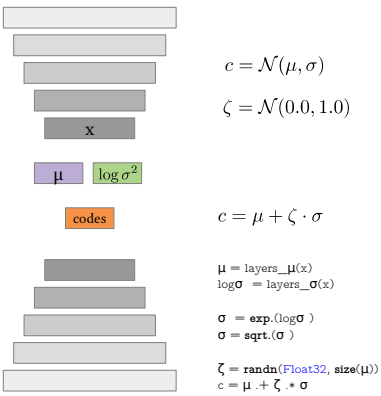


und so kann man sich das vorstellen





Backpropagation ist aber nur mit Reparametrisierung möglich!



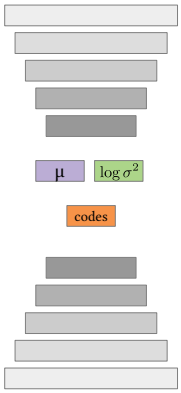
$c = \mathcal{N}(\mu, \sigma)$

$\zeta = \mathcal{N}(0.0, 1.0)$

$c = \mu + \zeta \cdot \sigma$

```
μ = layers_μ(x)
logσ  = layers_σ(x)

σ  = exp.(logσ )
σ = sqrt.(σ )

ζ = randn(Float32, size(μ))
c = μ .+ ζ .* σ
```

Die zusätzliche Lossfunktion wirkt wie ein Gummizug, der die Verteilungen zusammenhält:



Kullback–Leibler-Divergenz misst (hier) die Abweichung von der Standardnormalverteilung:

$$\mathcal{L}^{KL} = \frac{1}{2}\sum_i (\sigma_i^2 + \mu_i^2 - \ln \sigma_i^2 - 1)$$

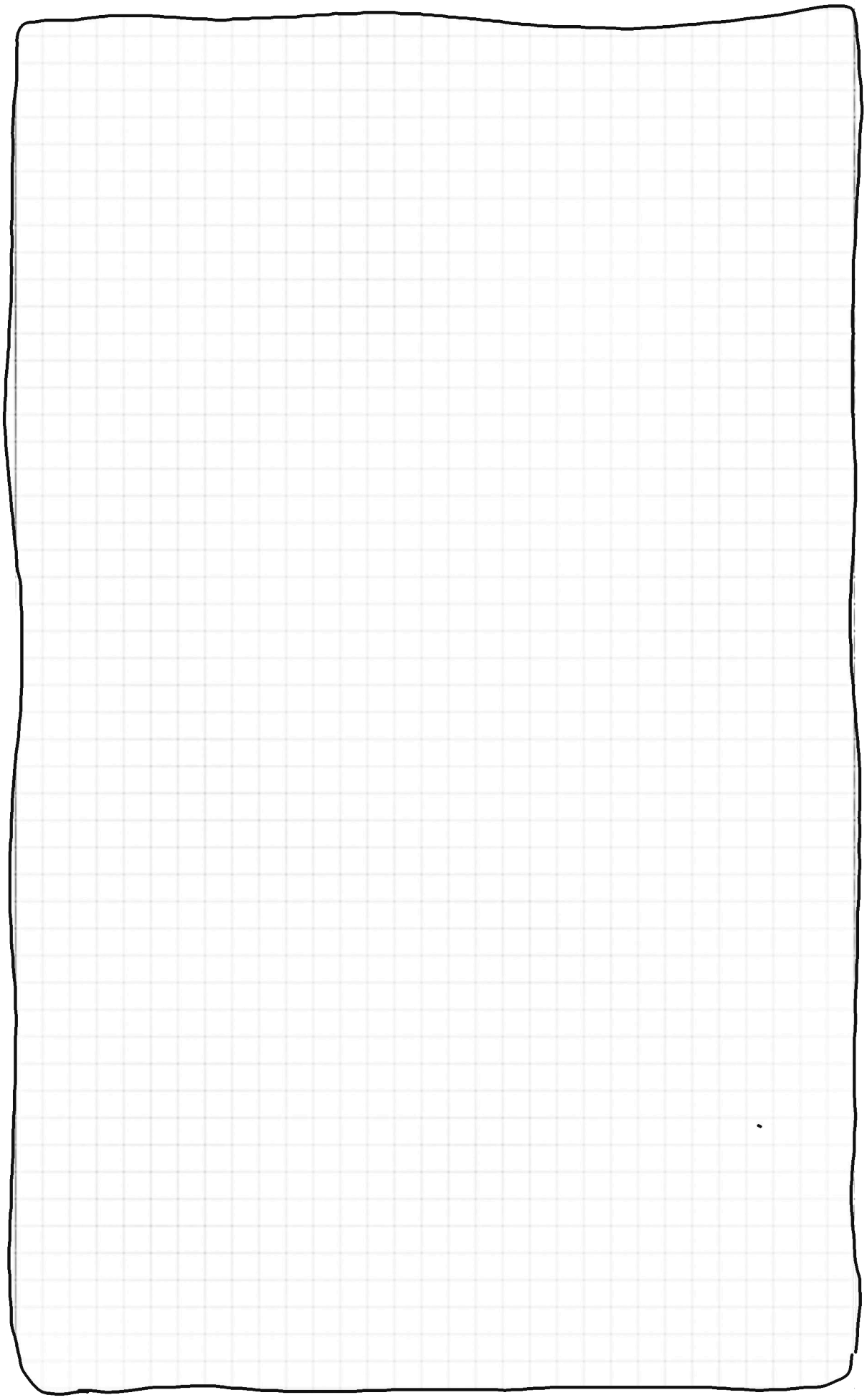
$$\mathcal{L} = \mathcal{L}^2 + \mathcal{L}^{KL}$$

```
n = length(x)
loss_2 = sum(abs2, x .- y) / n
loss_KL = sum( σ  .+ μ.*μ .- logσ  .- 1) / (2n)
loss = loss_2 + loss_KL
```

A. Asperti, M. Trentin, *Balancing reconstruction error and Kullback-Leibler divergence in Variational Autoencoders*, arXiv:2002.07514 [cs.NE].

Tutorial: so wird's umgesetzt

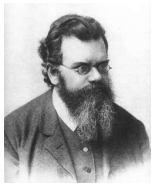
Code:



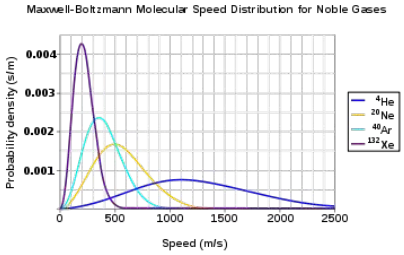
Übersicht. wie lassen sich AEs tief machen?

- Regularisierung → VAE
- **Tiefe → DBN**
- MLP → CNN-AE
- Informationsverlust im Bottleneck → uNet
- MLP → Seq2Seq

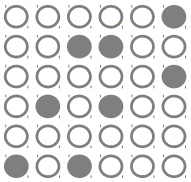
10.3 Restricted Boltzmann Machine (RBM)



$$N_j = N_0 \cdot g_j \cdot e^{-\frac{E_j}{k_B \cdot T}}$$



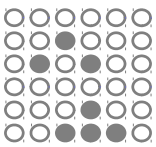
Boltzmann Machine (Hinton 1985)

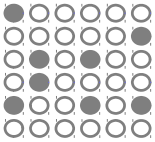


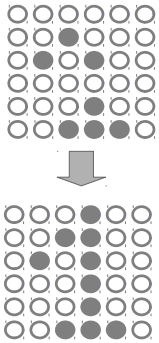
- "Energie:"

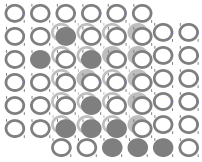
$$E = -(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i)$$

BM als assoziativer Speicher

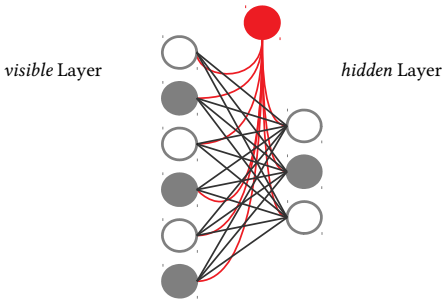








Restricted Boltzmann Machine (RBM, Paul Smolensky 1986)



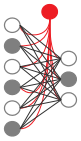
$$p(h_j = 1) = \sigma(\sum_i v_i w_{ij} + b_j)$$

$$p(v_i = 1) = \sigma(\sum_j h_j w_{ij} + a_i)$$

Energie eines Musters in der RBM

"Energie:"

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{ivisible} a_i v_i - \sum_{jhidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$



Wahrscheinlichkeit für Muster p:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})}$$

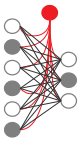
$$Z = \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$$

Loss in der RBM

Loss für Muster p:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$$

$$\log p(\mathbf{v}) \longrightarrow \Delta w_{ij} = \eta \cdot \frac{\partial \log p(\mathbf{v})}{\partial w_{ij}}$$

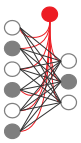


$$\frac{\partial \log p(\mathbf{v})}{\partial w_{ij}} = \langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{reconstruct}}$$

Simmulated Annealing hilft dem Optimierer

Loss für Muster p:

$$p(\mathbf{v}) = \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$$

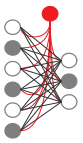


$$p(h_j = 1) = \sigma(\sum_i v_i w_{ij} + b_j)$$

$$p(v_i = 1) = \sigma(\sum_j h_j w_{ij} + a_i)$$

$$\sigma = \frac{1}{1 + e^{\frac{x}{T}}}$$

Training der RBM



- Input an v anlegen
- Propagieren nach h (binäre Zustände)

$$p(h_j = 1) = \sigma(\sum_i v_i w_{ij} + b_j)$$

- Propagieren zurück nach v

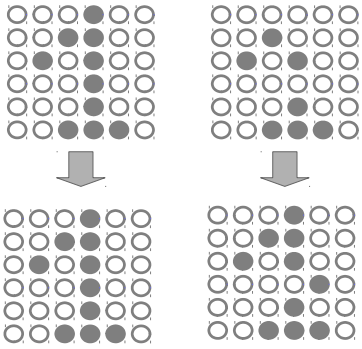
$$p(v_i = 1) = \sigma(\sum_j h_j w_{ij} + a_i)$$

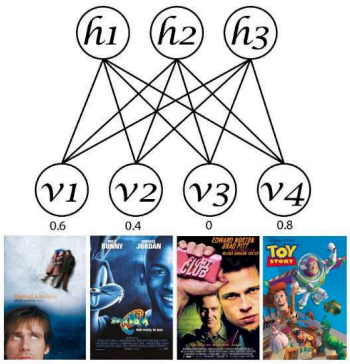
- Training

$$\Delta w_{ij} = \eta \cdot (\langle v_i h_j \rangle_{\text{data}} - \langle v_i h_j \rangle_{\text{reconstruct}})$$

- → bis selbstkonsistent
- → simulated annealing

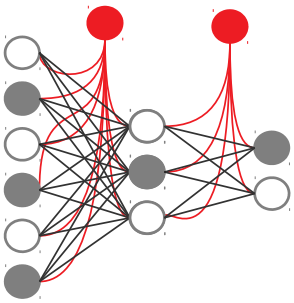
Beispiel: Assoziativer Speicher mit fuzzy Eigenschaften

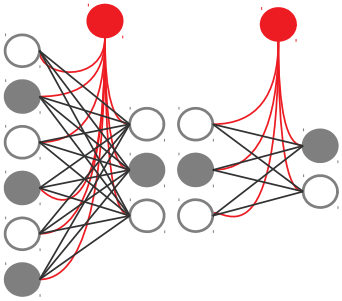




Mohammad Fawaz Siddiqi
Build a recommendation engine with a restricted Boltzmann machine using TensorFlow
https://developer.ibm.com/technologies/deep-learning/tutorials/build-a-recommendation-engine-with-a-restricted-boltzmann-machine-using-tensorflow/

Mehrschichtige RBM





10.4 Deep Believe Network

Schritt 1: RBM-Training

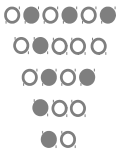
RBM: 20 000 ⟶ 1024

RBM: 1024 ⟶ 512

RBM: 512 ⟶ 64

RBM: 64 ⟶ 16

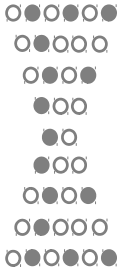
Schritt 2: Unrollung zum MLP



Dense: 20 000 ⟶ 1024
Dense: 1024 ⟶ 512
Dense: 512 ⟶ 64
Dense: 64 ⟶ 16

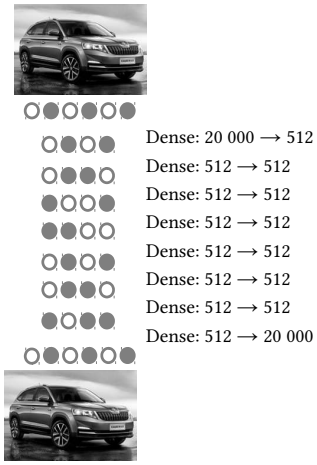
Kategorien

oder: Unrolling zum AE



Dense: 20 000 ⟶ 1024
Dense: 1024 ⟶ 512
Dense: 512 ⟶ 64
Dense: 64 ⟶ 16
Dense: 16 ⟶ 64
Dense: 64 ⟶ 512
Dense: 512 ⟶ 1024
Dense: 1024 ⟶ 20 000

geht sehr *deep* und *wide*

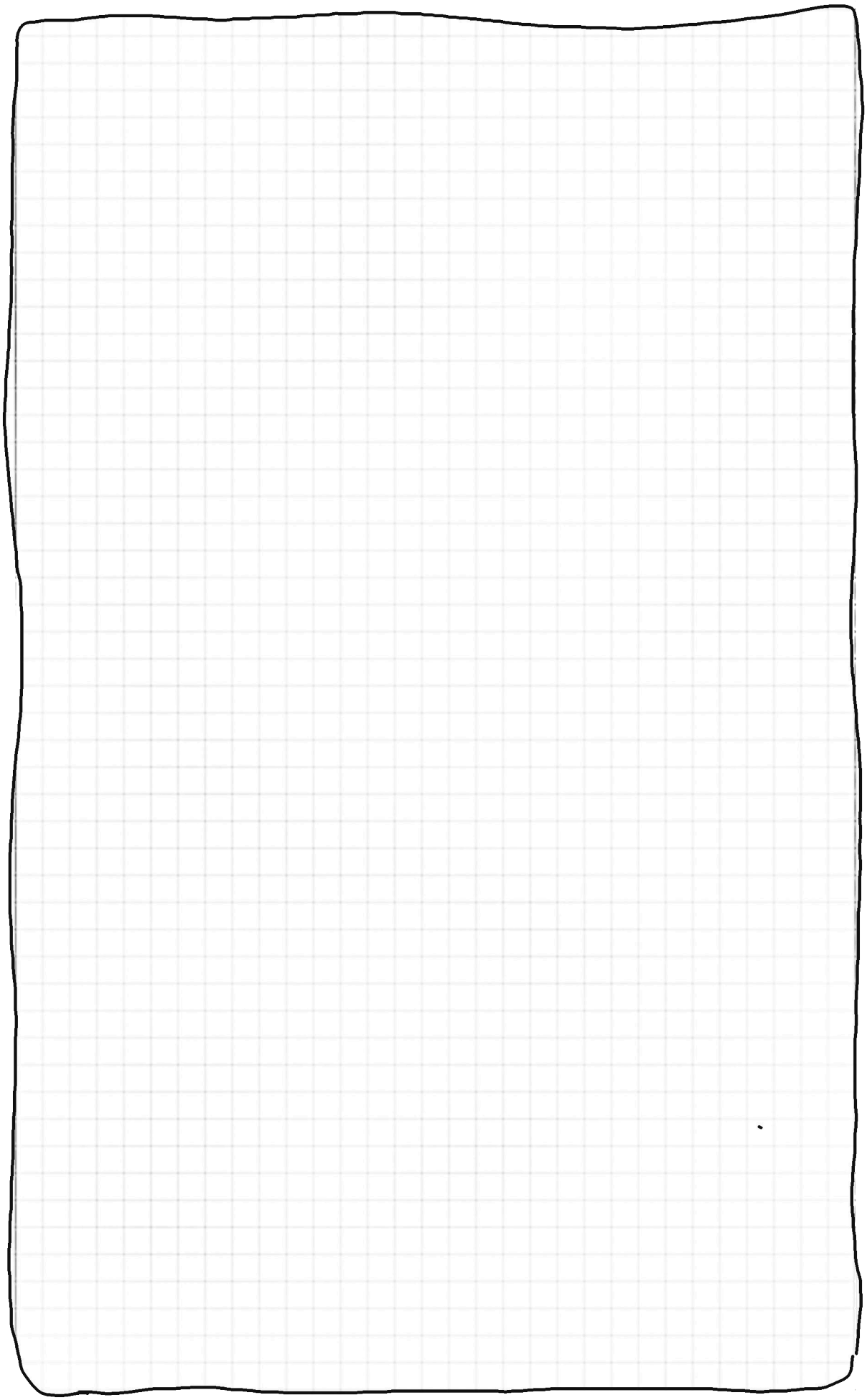


10.5 CNN-Autoencoder

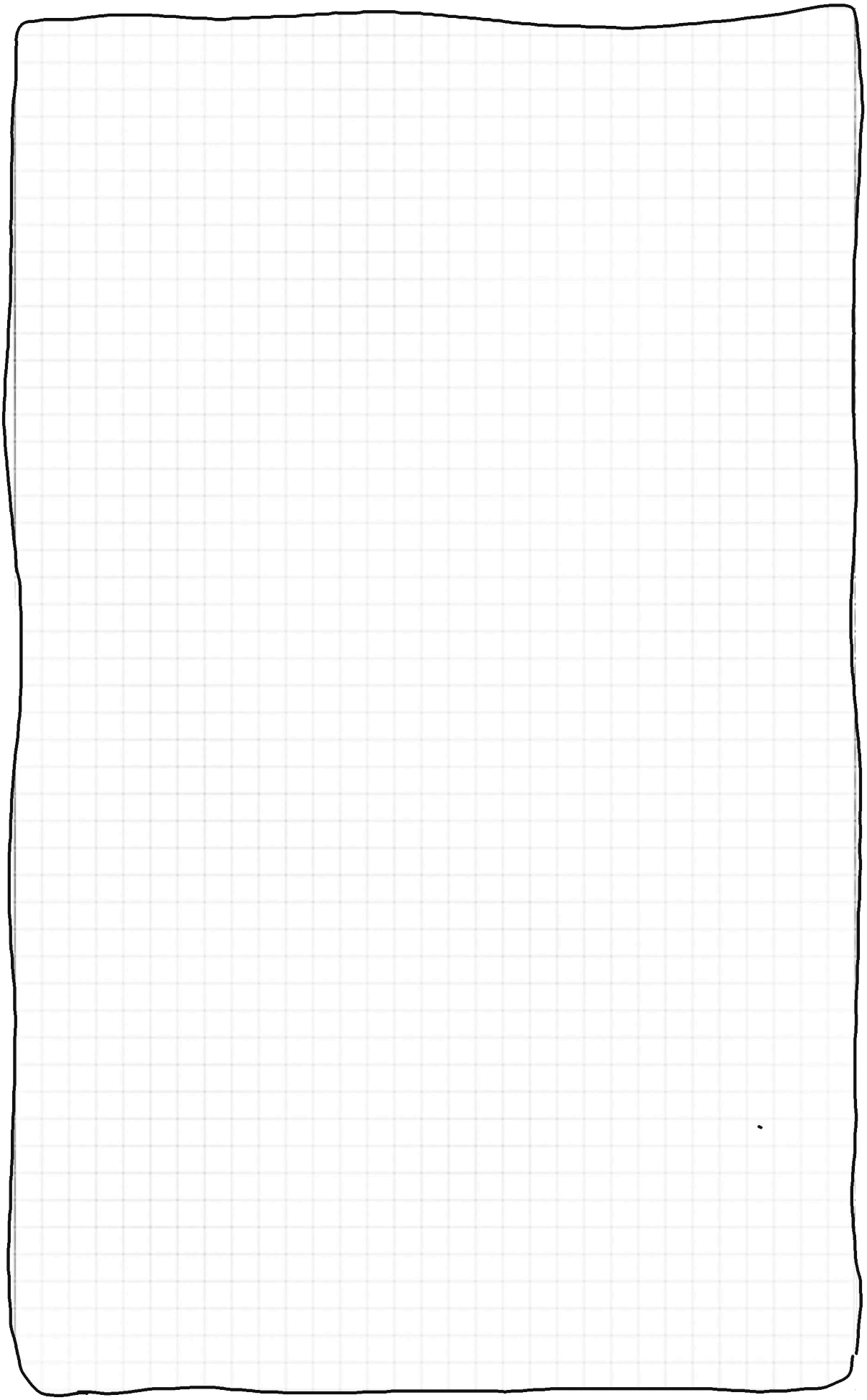
Übersicht

- Regularisierung → VAE
- Tiefe → DBN
- **MLP → CNN-AE**
- Informationsverlust im Bottleneck → uNet
- MLP → Seq2Seq

Convolution und Deconvolution

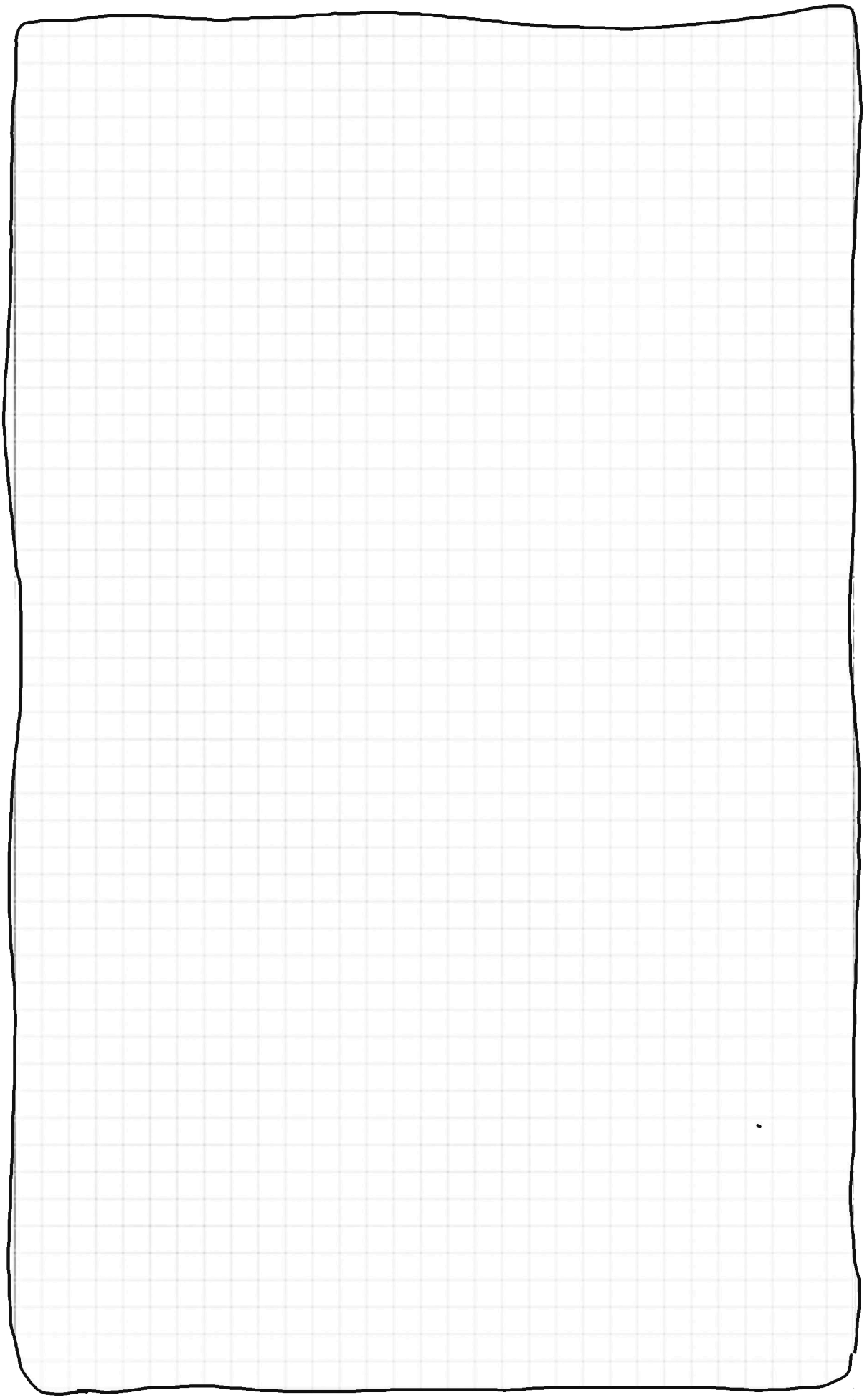


Unpooling und Deconvolution mit Stride 2



Tutorial: so wird's umgesetzt

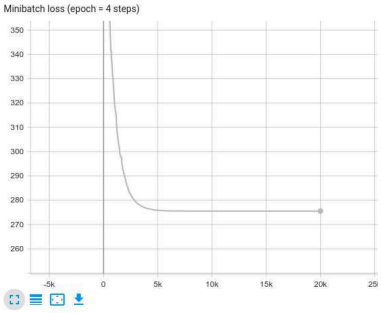
Code:



1. Versuch: stumpf

```
actf = relu
encoder = Chain(
        Conv(5,5,3,32; padding=2, actf=actf),
        Pool(),
        Conv(5,5,32,16; padding=2, actf=actf),
        Pool(),
        Conv(5,5,16,8; padding=2, actf=actf),
        Pool()
)

decoder = Chain(
        DeConv(5,5,8,16; stride=2, padding=0, actf=actf),
        DeConv(5,5,16,32; stride=2, padding=0, actf=actf),
        DeConv(5,5,32,3; stride=2, padding=0, actf=actf),
        x->crop_array(x, (64,64))
)

md1 = Regressor(encoder, decoder)
```

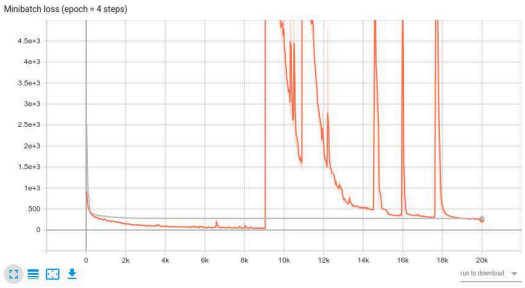
Flat Spot!



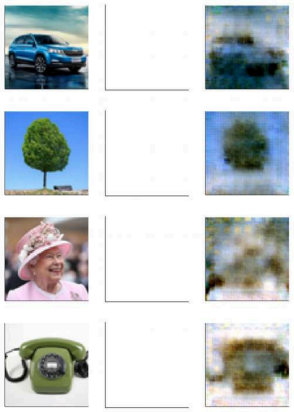
2. Versuch: Leaky

```
leaky(x) = relu(x) + Float32(0.01)*x
actf = leaky
encoder = Chain(
        Conv(5,5,3,32; padding=2, actf=actf),
        Pool(),
        Conv(5,5,32,16; padding=2, actf=actf),
        Pool(),
        Conv(5,5,16,8; padding=2, actf=actf),
        Pool()
)

decoder = Chain(
        DeConv(5,5,8,16; stride=2, padding=0, actf=actf),
        DeConv(5,5,16,32; stride=2, padding=0, actf=actf),
        DeConv(5,5,32,3; stride=2, padding=0, actf=actf),
        x->crop_array(x, (64,64))
)

md1 = Regressor(encoder, decoder)
```

Besser, aber...

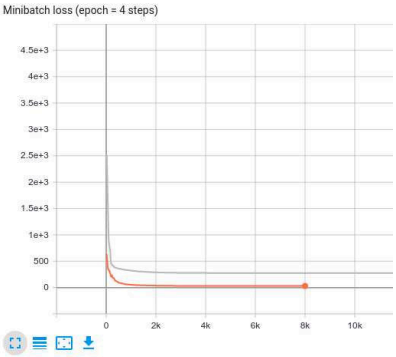


eraly Stopping hilft, ist aber nicht die Lösung!

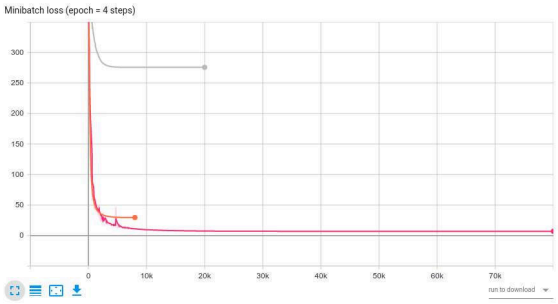


3. Versuch: Batch-Normalisation ist notwendig:

```
leaky(x) = relu(x) + Float32(0.01)*x
actf = leaky
encoder = Chain(
        Conv(5,5,3,32; padding=2, actf=actf),
        Pool(),
        Conv(5,5,32,16; padding=2, actf=actf),
        Pool(),
        Conv(5,5,16,8; padding=2, actf=actf),
        Pool(),
        BatchNorm()
)

decoder = Chain(
        DeConv(5,5,8,16; stride=2, padding=0, actf=actf),
        DeConv(5,5,16,32; stride=2, padding=0, actf=actf),
        DeConv(5,5,32,3; stride=2, padding=0, actf=actf),
        x->crop_array(x, (64,64))
)

mdl = Regressor(encoder, decoder)
```

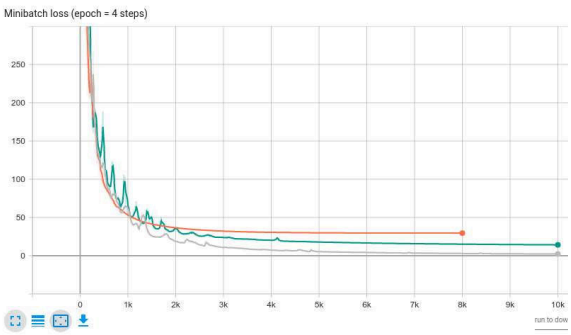


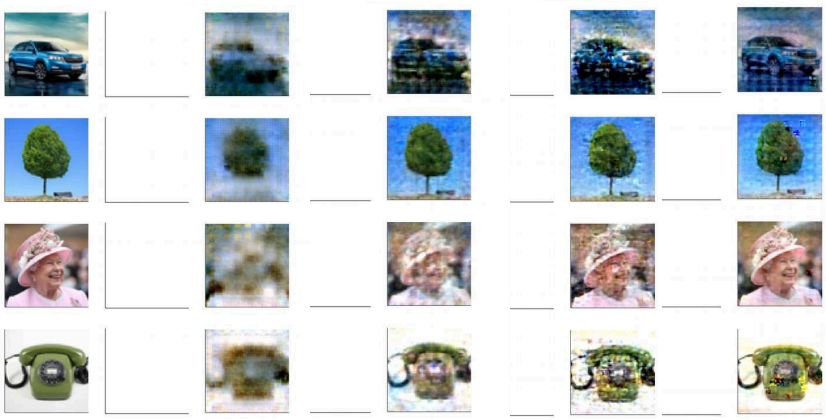
4. Versuch: Darf es etwas mehr sein?



5. Versuch: Ultima Ratio

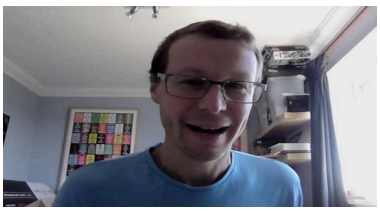
```
leaky(x) = relu(x) + Float32(0.01)*x
actf = leaky
encoder = Chain(
          Conv(5,5,3,32; padding=2, actf=actf),
          Pool(),
          Conv(5,5,32,64; padding=2, actf=actf),
          Pool(),
          Conv(5,5,64,128; padding=2, actf=actf),
          Pool(),
          BatchNorm()
)

decoder = Chain(
          DeConv(5,5,128,64; stride=2, padding=0, actf=actf),
          DeConv(5,5,64,32; stride=2, padding=0, actf=actf),
          DeConv(5,5,32,3; stride=2, padding=0, actf=actf),
          x->crop_array(x, (64,64))
)

mdl = Regressor(encoder, decoder)
```





Anwendungen

Bildsegmentierung

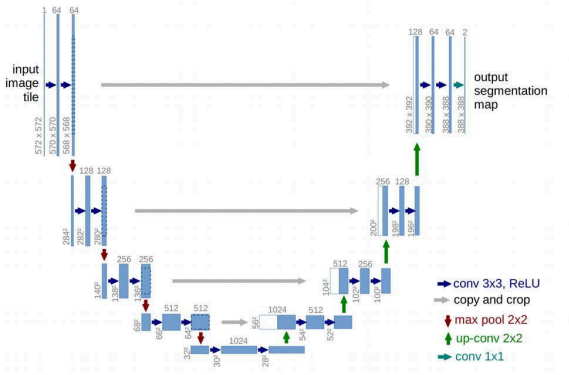


10.6 uNet

Übersicht

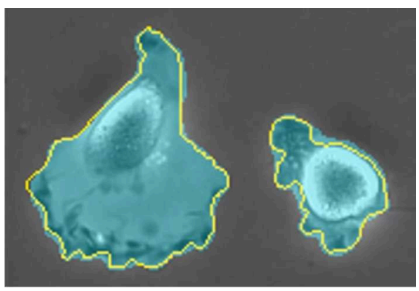
- Regularisierung → VAE
- Tiefe → DBN
- MLP → CNN-AE
- **Informationsverlust im Bottleneck → uNet**
- MLP → Seq2Seq

Das uNet

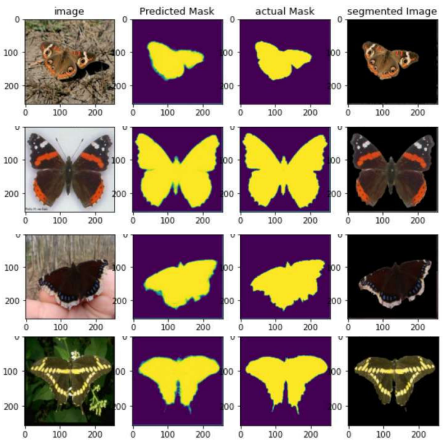


O. Ronneberger, P. Fischer, Th. Brox, *U-Net: Convolutional Networks for Biomedical Image Segmentation*, arXiv:1505.04597 [cs.CV]

uNet Anwendungen: Biomedical Image Segmentation



uNet Anwendungen: Butterfly



Coding U-Net For Image Segmentation
Butterfly Dataset
https://www.kaggle.com/vanvalkenberg/coding-u-net-for-image-segmentation

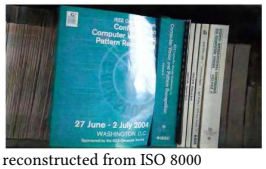
uNet Anwendungen: Superresolution

Super Resolution: Low Res to High Res Images
Super Image Resolution
https://www.kaggle.com/vanvalkenberg/
super-resolution-low-res-to-high-res-images

uNet Anwendungen: See in the Dark

ISO 8000

ISO 409600



reconstructed from ISO 8000

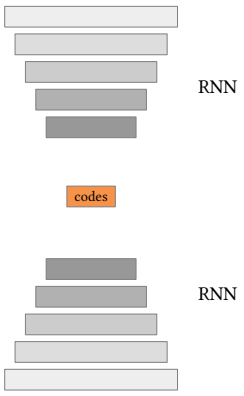
C. Chen, Q. Chen, J. Xu, V. Koltun, *Learning to See in the Dark*,
arXiv:1805.01934v1 [cs.CV]

10.7 Sequence-2-Sequence Networks

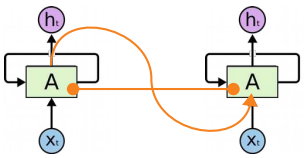
Übersicht

- Regularisierung → VAE
- Tiefe → DBN
- MLP → CNN-AE
- Informationsverlust im Bottleneck → uNet
- **MLP → Seq2Seq**

Geht auch mit RNNs



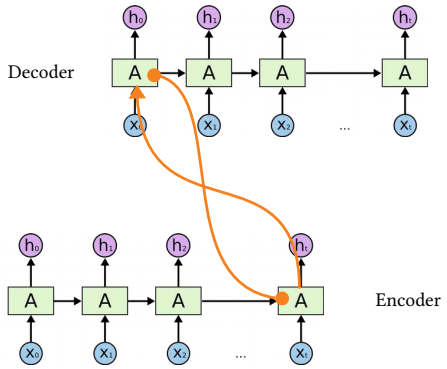
Im Bildchen von Chris Olah:



Encoder Decoder

colah's Blog:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unrolling nach Chris Olah:



colah's Blog:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Die Darstellung von Google ist genauer?

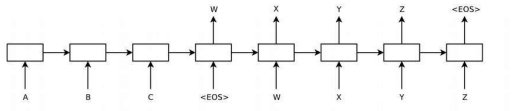
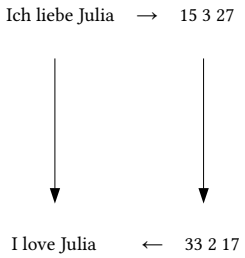


Figure 1: Our model reads an input sentence "ABC" and produces "WXYZ" as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.

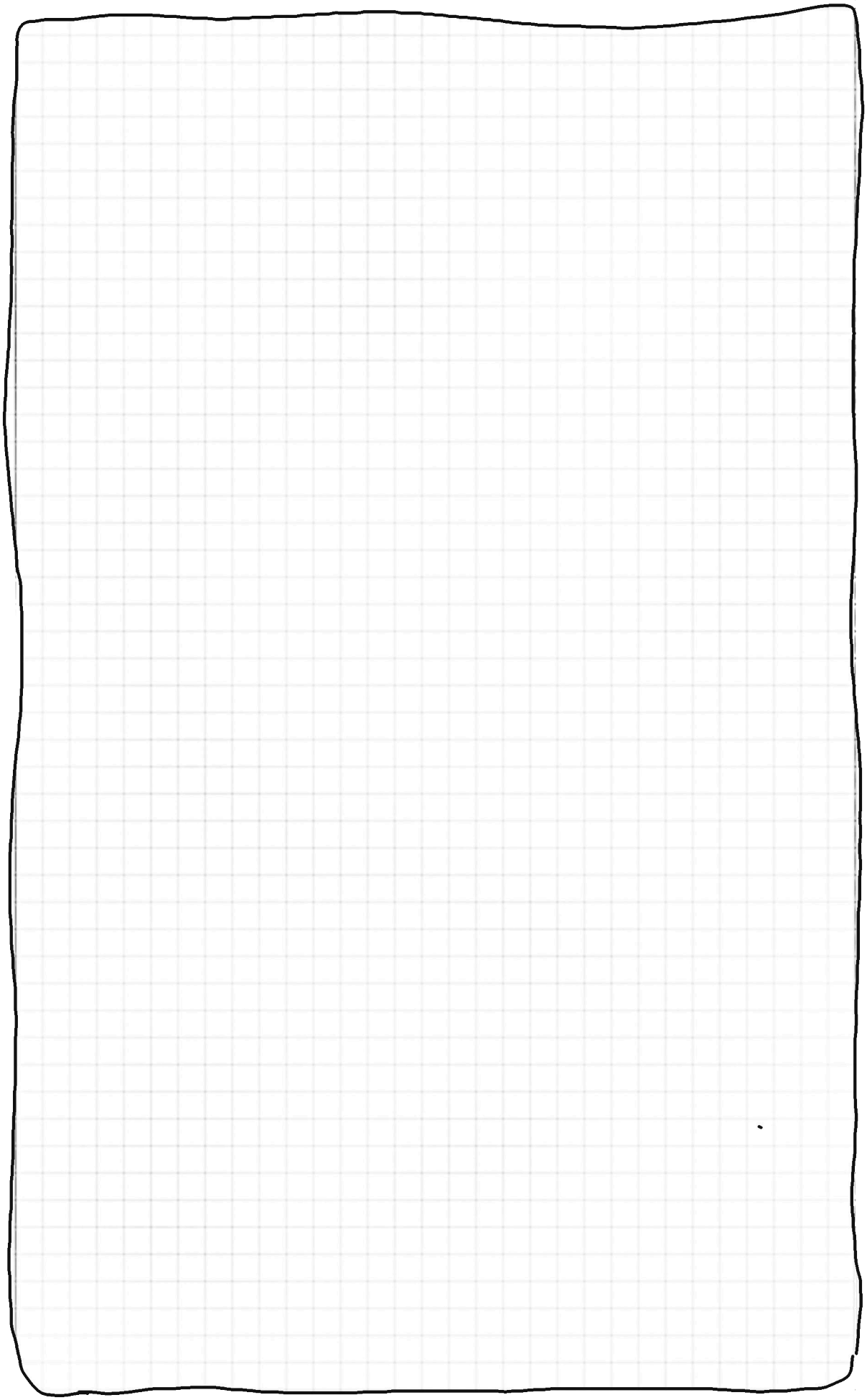
I. Sutskever; O. Vinyals, V.Le Quoc; *Sequence to Sequence Learning with Neural Networks.*
arXiv:1409.3215 [cs] 2014.

Implementierung des Übersetzers als Seq2Seq-Netzwerk

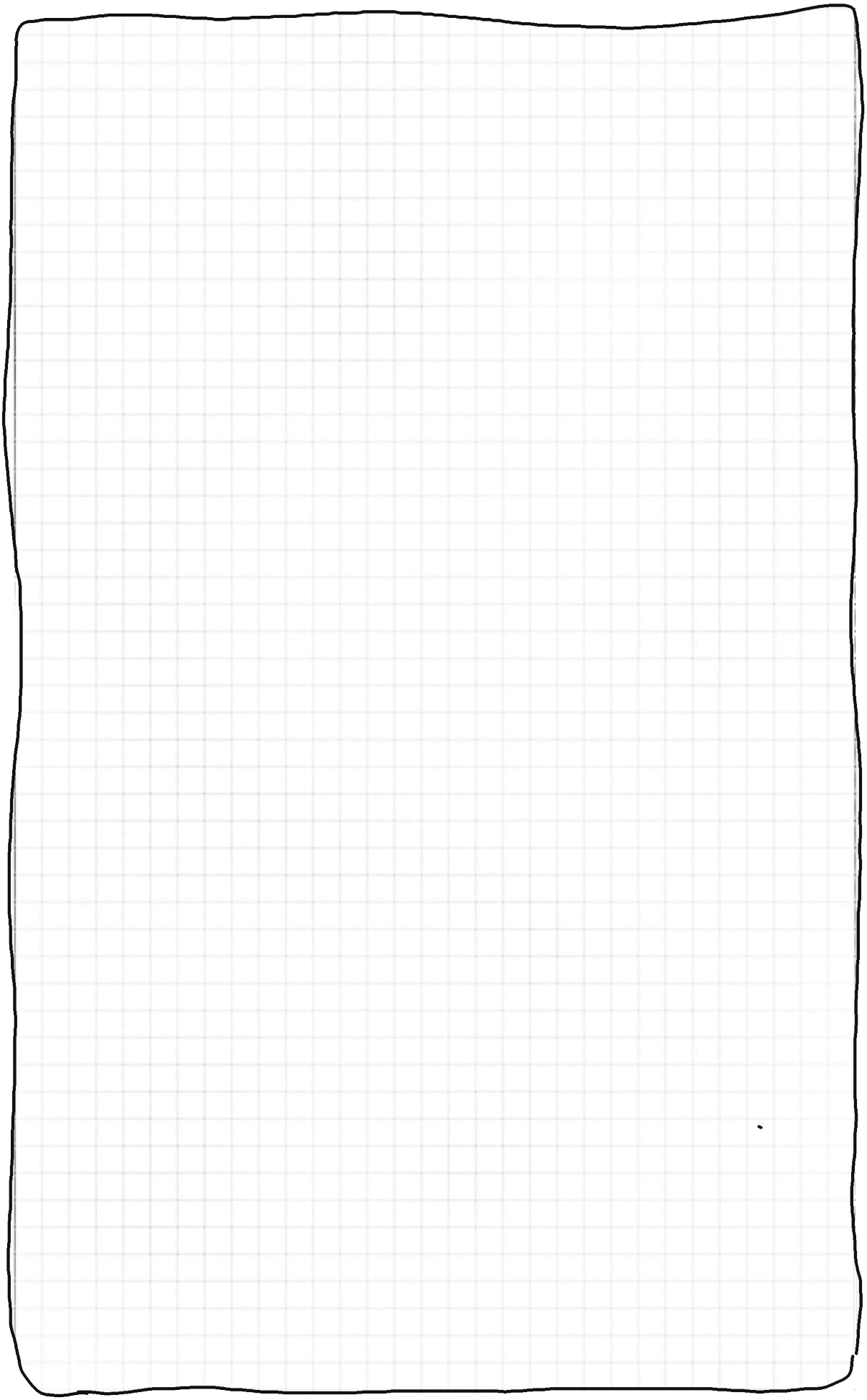
Ganz genau am Beispiel eines Übersetzers, der Sequenzen von Wörtern auf Wörter mapped:



Schemazeichnung:



Implementierung:



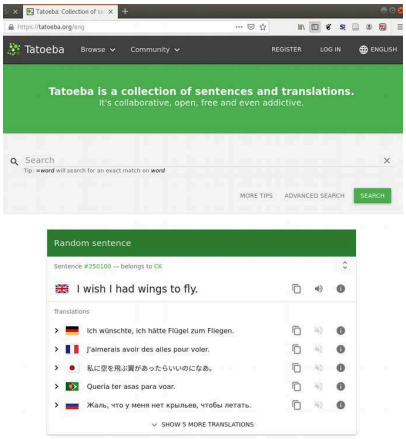
Was noch bedacht werden muss...

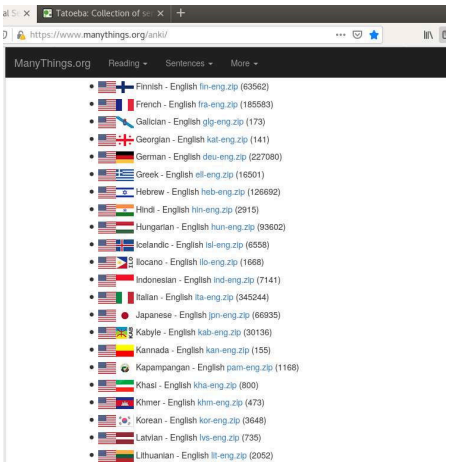
- Datensatz
- Regularisierung
- Geduld

Tipps:

- Reverse/bidirectional
- Minibatches

Tatoeba Datensatz:





Weitere Datenquellen

- EU-Bürokratie
- Mozilla
- ...

Import geht einfach

In [42]:
```
using Unicode
data_file = "Data/deu.txt"
data = readlines(data_file)
is, os = [], []
for line in data
    en, de, attr = split(line, "\t")

    de = Unicode.normalize(de)
    de = replace(de, Regex("[.!?,;:]") => "")

    en = Unicode.normalize(en)
    en = replace(en, Regex("[.!?,;:\"\']") => "")
    push!(is, de)
    push!(os, en)
end
is
```

Out[42]:
```
227080-element Array{Any,1}:
 "Geh"
 "Hallo"
 "Grüß Gott"
 "Lauf"
 "Lauf"
 "Potzdonner"
 "Donnerwetter"
 "Feuer"
 "Hilfe"
 "Zu Hülf"
 "Stopp"
 "Anhalten"
 "Warte"
 ⋮
 "Ich empfehle muttersprachliche Sätze beizutragen denn bei diesen kann man au
klingen"
 "Ein Gebäude mit hohen Decken und riesigen Räumen mag weniger praktisch sein
setzt aber dafür fügt es sich oft gut in seine Umgebung ein"
 "Als Streich ließen einige Schüler drei Ziegen in ihrer Schule los nachdem si
en der Ziegen gemalt hatten Die Lehrer brachten den Großteil des Tages damit z
```

Regularisierung mit etwas dropout() ist immer möglich:

Model:

:
```
mutable struct S2S
    embed_enc
    drop
    embed_dec
    lstm_enc
    lstm_dec
    predict
    voc_in; voc_out; embed; units
end

function S2S(n_embed, n_units, n_vocab_in, n_vocab_out)
    embed_enc = Embed(n_vocab_in, n_embed)
    drop = Dropout(0.1)
    embed_dec = Embed(n_vocab_out, n_embed)
    lstm_enc = LSTM(n_embed, n_units)
    lstm_dec = LSTM(n_embed, n_units)
    predict = Predictions(n_units, n_vocab_out)

    S2S(embed_enc, drop, embed_dec, lstm_enc, lstm_dec, predict,
        n_vocab_in, n_vocab_out, n_embed, n_units)
end
```

Dann ist Geduld nötig!

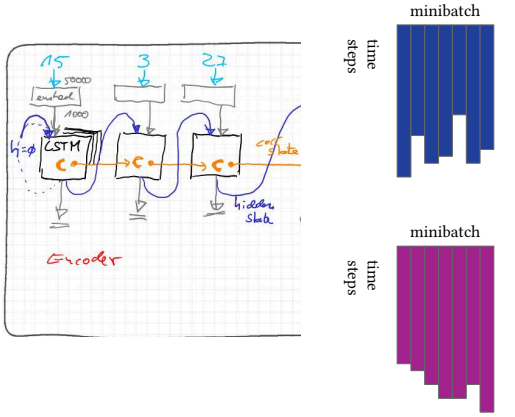
Ein weiterer Tipp: Rückwärts trainieren ist manchmal besser.

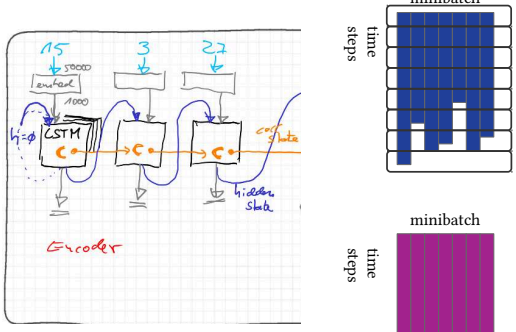
```
Ich liebe Julia -> I love Julia
```

```
Julia liebe Ich -> I love Julia
```

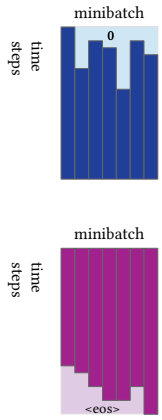
Minibatches bei RNNs

Problem: die Sequenzen sind nicht gleich lang!

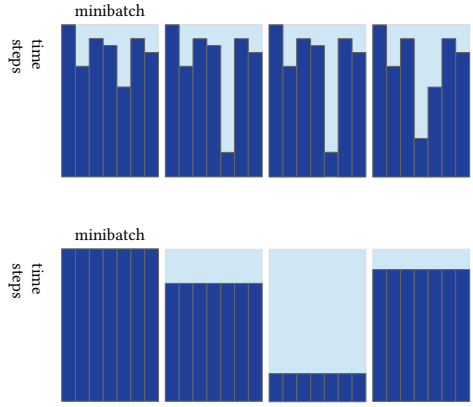




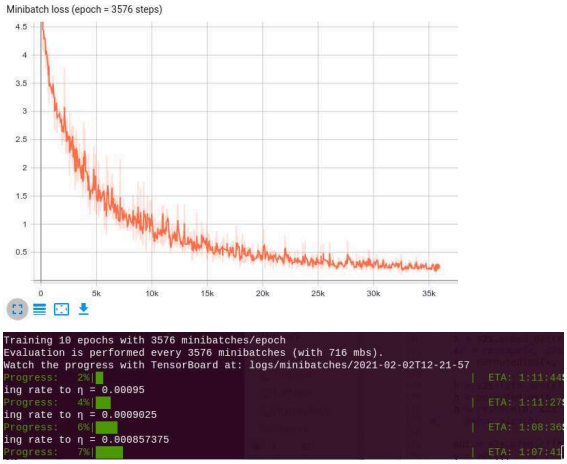
Padding vorne oder hinten; je nachdem, was sinnvoller ist:

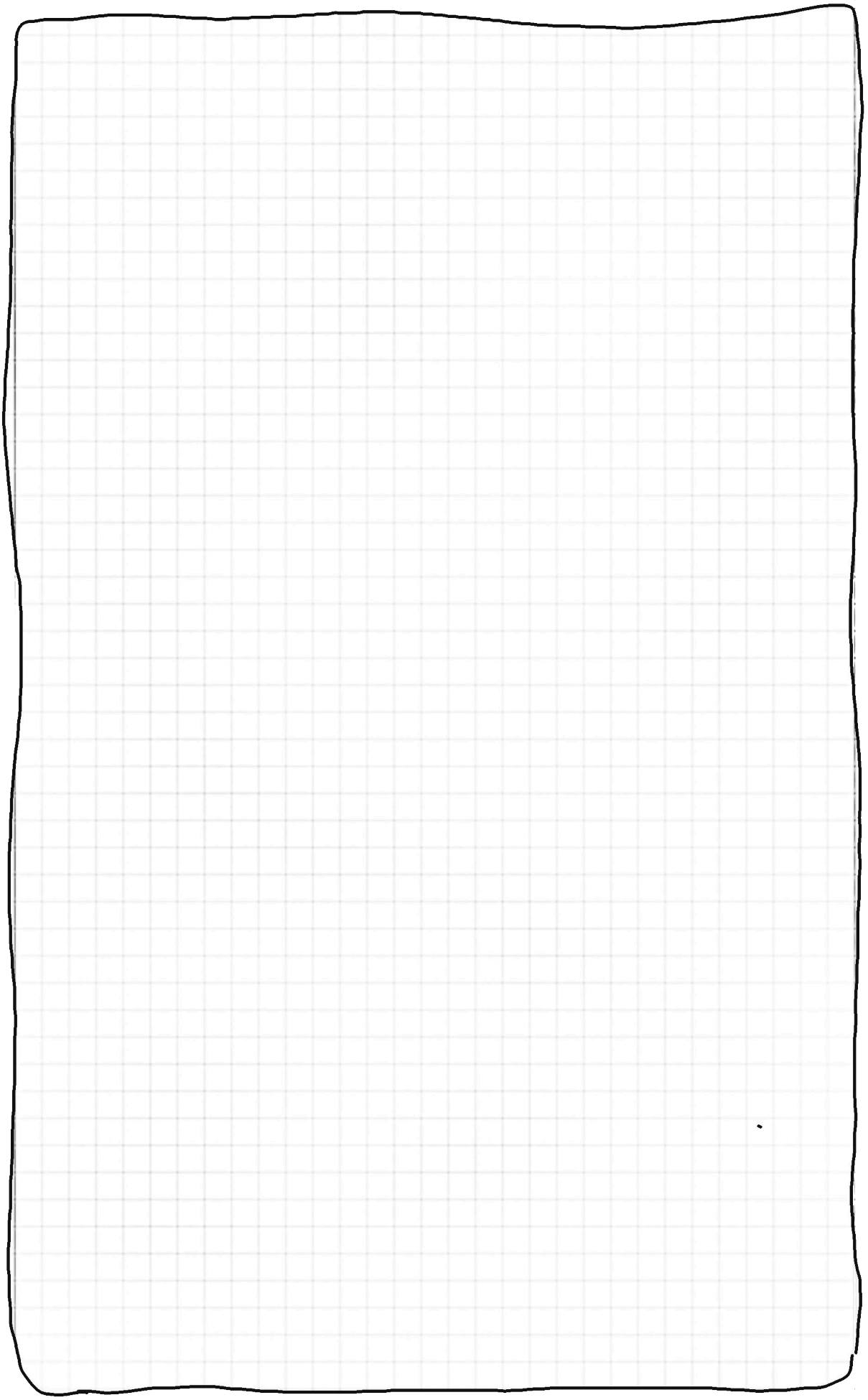


Besser als Padding ist sortieren! Ziel: innerhalb jeder Minibatch müssen die Sequenzen gleich lang sein.



Mit mb-size=64: 7 min pro Epoche statt 7 Stunden!



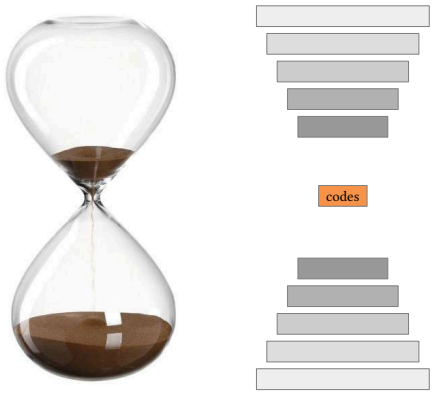


Kapitel 11:

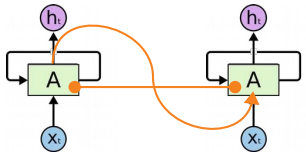
Attention-Mechanismen

11.1 Was ist Attention?

Recap: Seq2Seq



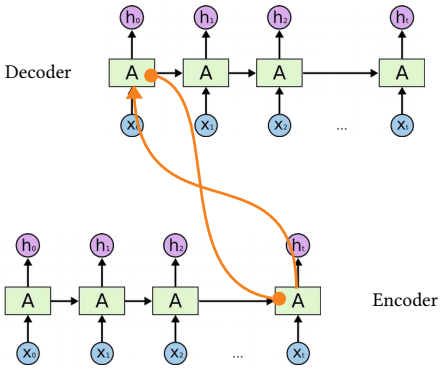
s2s in den Bildchen von Chris Olah



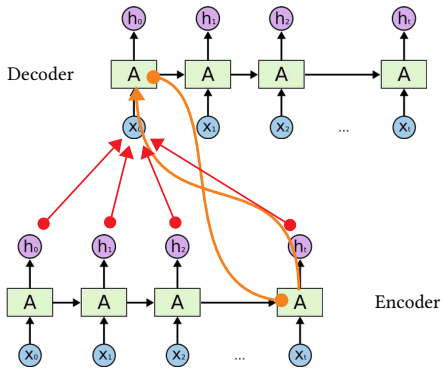
Encoder Decoder

colah's Blog:
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

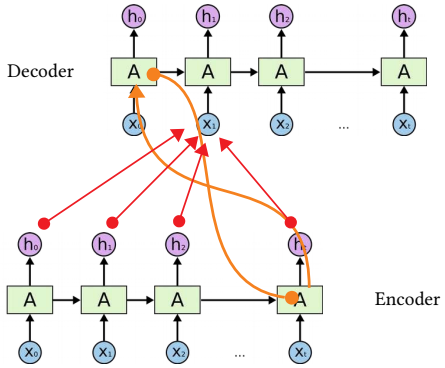
s2s unrolled ...



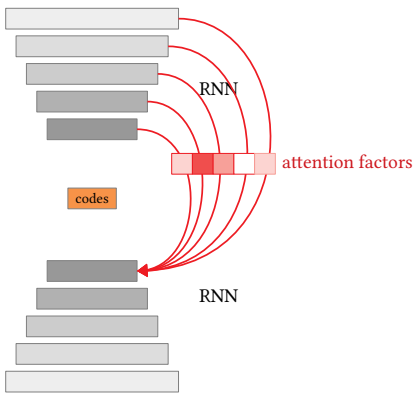
Ohne Bottleneck - Step 1



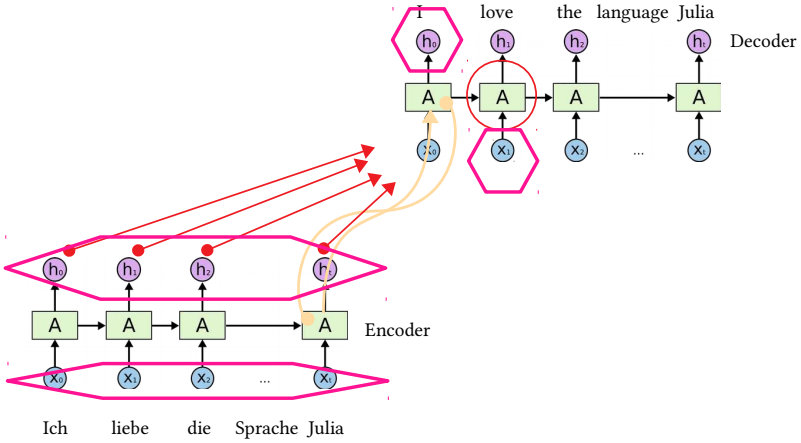
Ohne Bottleneck - Step 2



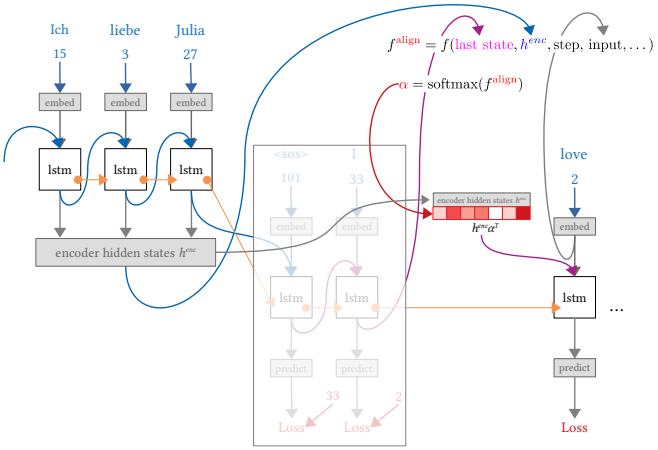
Besser mit Attention Factors α



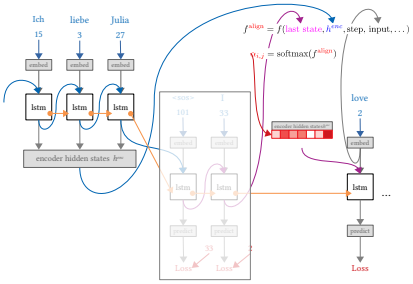
Alles kann die Attention beeinflussen:



in unserer Darstellung:



Berechnung der Alignment Scores, der Attention Factors und des neuen Context Vectors, der den alten (== hidden State (t-1) ersetzt:

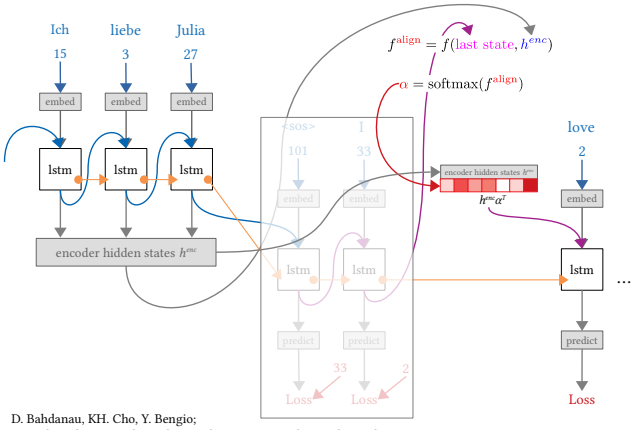


$$f^{\text{align}} = f(\text{last state}, h^{enc}, \text{step, input}, \dots)$$

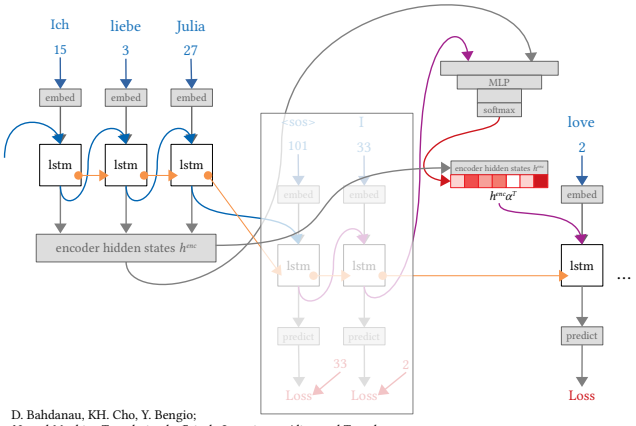
$$\alpha_{i,j} = \text{softmax}(f^{\text{align}}(\dots))$$

$$c_j = \sum_{i=1}^{t} \alpha_i h_{ij}^{enc} = h^{enc} \cdot \alpha_i^T$$

11.2 Bahdanau-Attention (concat/additive):



D. Bahdanau, KH. Cho, Y. Bengio;
Neural Machine Translation by Jointly Learning to Align and Translate
ICLR 2015.



D. Bahdanau, KH. Cho, Y. Bengio;
Neural Machine Translation by Jointly Learning to Align and Translate
ICLR 2015.

Umsetzung:

4.1 Dataset

WMT '14 contains the following English-French parallel corpora: Europarl (61M words), news commentary (5.5M), UN (421M) and two crawled corpora of 90M and 272.5M words respectively, totaling 850M words. Following the procedure described in Cho *et al.* (2014a), we reduce the size of the combined corpus to have 348M words using the data selection method by Axelrod *et al.* (2011).[5] We do not use any monolingual data other than the mentioned parallel corpora, although it may be possible to use a much larger monolingual corpus to pretrain an encoder. We concatenate news-test-

4.2 Models

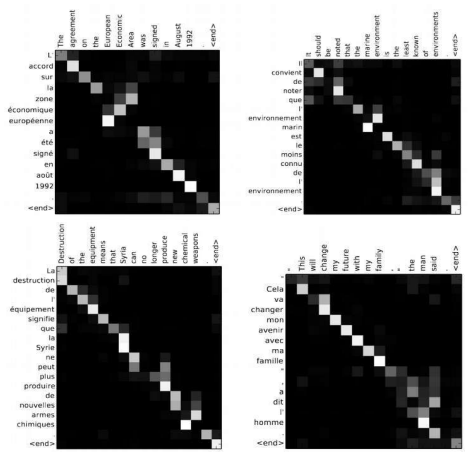
We train two types of models. The first one is an RNN Encoder–Decoder (RNNencdec, Cho *et al.*, 2014a), and the other is the proposed model, to which we refer as RNNsearch. We train each model twice: first with the sentences of length up to 30 words (RNNencdec-30, RNNsearch-30) and then with the sentences of length up to 50 word (RNNencdec-50, RNNsearch-50).

The encoder and decoder of the RNNencdec have 1000 hidden units each.[7] The encoder of the RNNsearch consists of forward and backward recurrent neural networks (RNN) each having 1000 hidden units. Its decoder has 1000 hidden units. In both cases, we use a multilayer network with a single maxout (Goodfellow *et al.*, 2013) hidden layer to compute the conditional probability of each target word (Pascanu *et al.*, 2014).

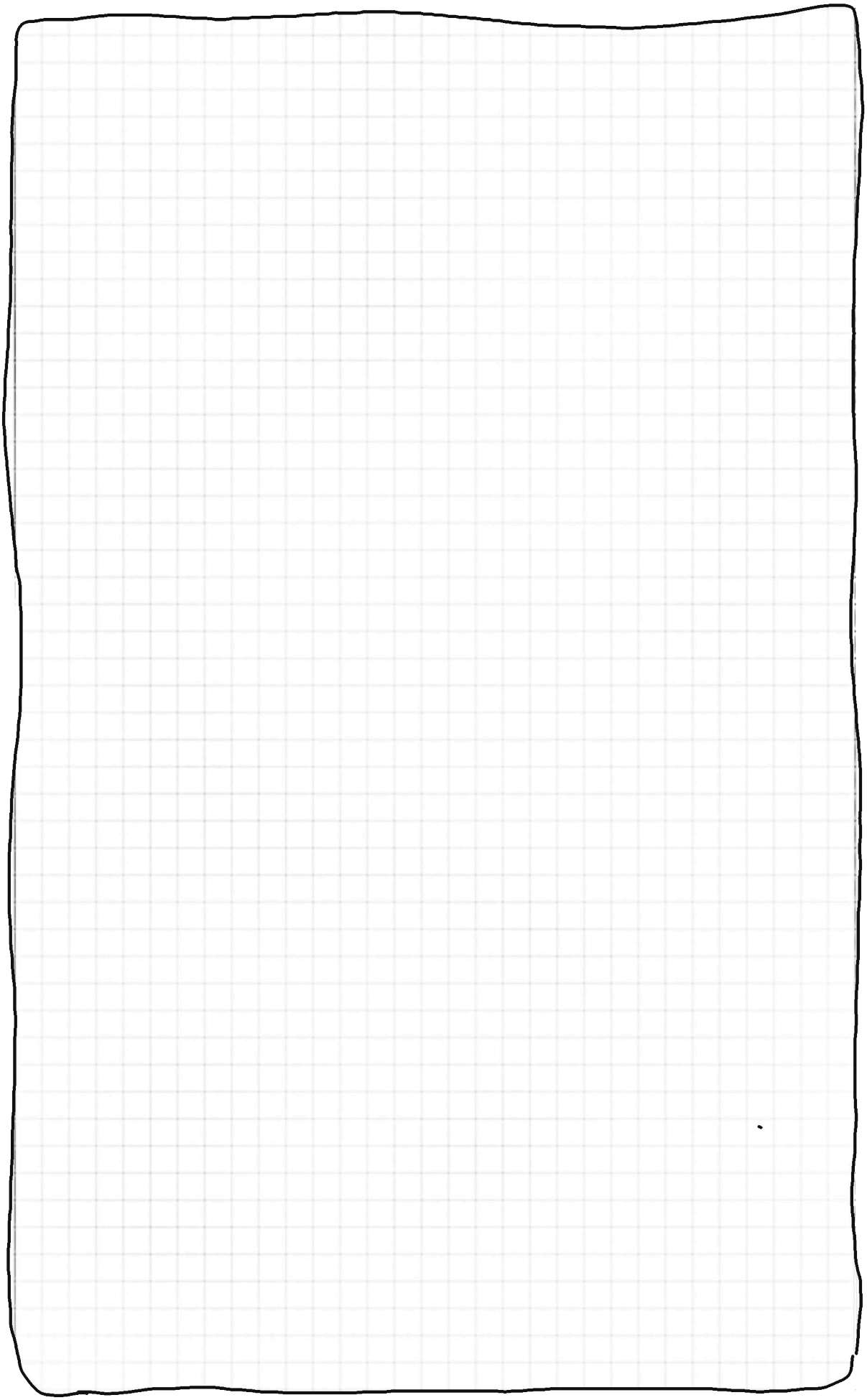
We use a minibatch stochastic gradient descent (SGD) algorithm together with Adadelta (Zeiler, 2012) to train each model. Each SGD update direction is computed using a minibatch of 80 sentences. We trained each model for approximately 5 days.

D. Bahdanau, KH. Cho, Y. Bengio;
Neural Machine Translation by Jointly Learning to Align and Translate
ICLR 2015.

Ergebnisse:


The agreement on the European Economic Area was signed in August 1992 <end>
L' accord sur la zone économique européenne a été signé en août 1992 <end>
It should be noted that the marine environment is the least known of environments <end>
Il convient de noter que l' environnement marin est le moins connu de l' environnement <end>
Destruction of the equipment means that Syria can no longer produce new chemical weapons <end>
La destruction de l' équipement signifie que la Syrie ne peut plus produire de nouvelles armes chimiques <end>
This will change my future with my family . the man said <end>
Cela va changer mon avenir avec ma famille . a dit l' homme <end>

Umsetzung des *additive-Style:*



Beispiel-Implementierung ohne Minibatches als mini-MLP:

$$f^{\text{align}} = w_{combine} \tanh(w_{dec} h_{t-1}^{dec} + w_{enc} h^{enc})$$

```
struct AttnBahdanau
    enc
    dec
    combine
    AttnBahdanau(n_units) = new(Dense(n_units, n_units, actf=identity),
                                Dense(n_units, n_units, actf=identity),
                                Dense(n_units, 1, actf=identity))
end
```

```
# simple: only w/o minibatch
#
function (attn::AttnBahdanau)(h_t, h_enc)   # h_t is a (n_units),
                                            # h_enc is (n_units, n_steps)

    steps = size(h_enc)[2]

    # expand h_t to fit all steps -> h_dec is (n_units, n_steps):
    #
    h_dec = hcat((h_t for i in 1:steps)...)
    attn_score = attn.combine(tanh.(attn.enc(h_enc) .+ attn.dec(h_dec)))
    α = softmax(attn_score)

    context = h_enc * permutedims(α)
    return context, α
end
```

```
a = AttnBahdanau(3)
a(h_t, h_enc)
```

Beispiel-Implementierung mit Minibatches:

$$f^{\text{align}} = w_{combine} \tanh(w_{dec} h_{t-1}^{dec} + w_{enc} h^{enc})$$

```
struct AttnBahdanau
    enc
    dec
    combine
    AttnBahdanau(n_units) = new(Dense(n_units, n_units, actf=identity),
                                Dense(n_units, n_units, actf=identity),
                                Dense(n_units, 1, actf=identity))
end
```

```
function (attn::AttnBahdanau)(h_t, h_enc)   # h_t is a (n_units, <n_mb>),
                                            # h_enc is (n_units, <n_mb>, n_steps)
    # make all 3d:
    #
    h_encR = reshape(h_enc, size(h_enc)[1], :, size(h_enc)[ndims(h_enc)])
    units, mb, steps = size(h_enc)
    h_tR = reshape(h_t, size(h_t)[1], :)

    # this is possible, because the TensorDense-layers are used:
    #
    attn_score = attn.combine(tanh.(attn.enc(h_encR) .+ attn.dec(h_tR)))
    α = softmax(attn_score, dims=3)

    # calc. context from encoder states:
    #
    c = sum(α .* h_encR, dims=3)

    # remove unneeded dims:
    #
    c = reshape(c, units, mb)
    α = reshape(α, mb, steps)

    return c, α
end
```

11.3 Luong-Attention (multiplicative/general)

Alignment-Funktionen aus dem Luong-Paper:

Alignment Score Function: $f^{\text{align}} = f(\text{last state}, \text{encoder annotations})$

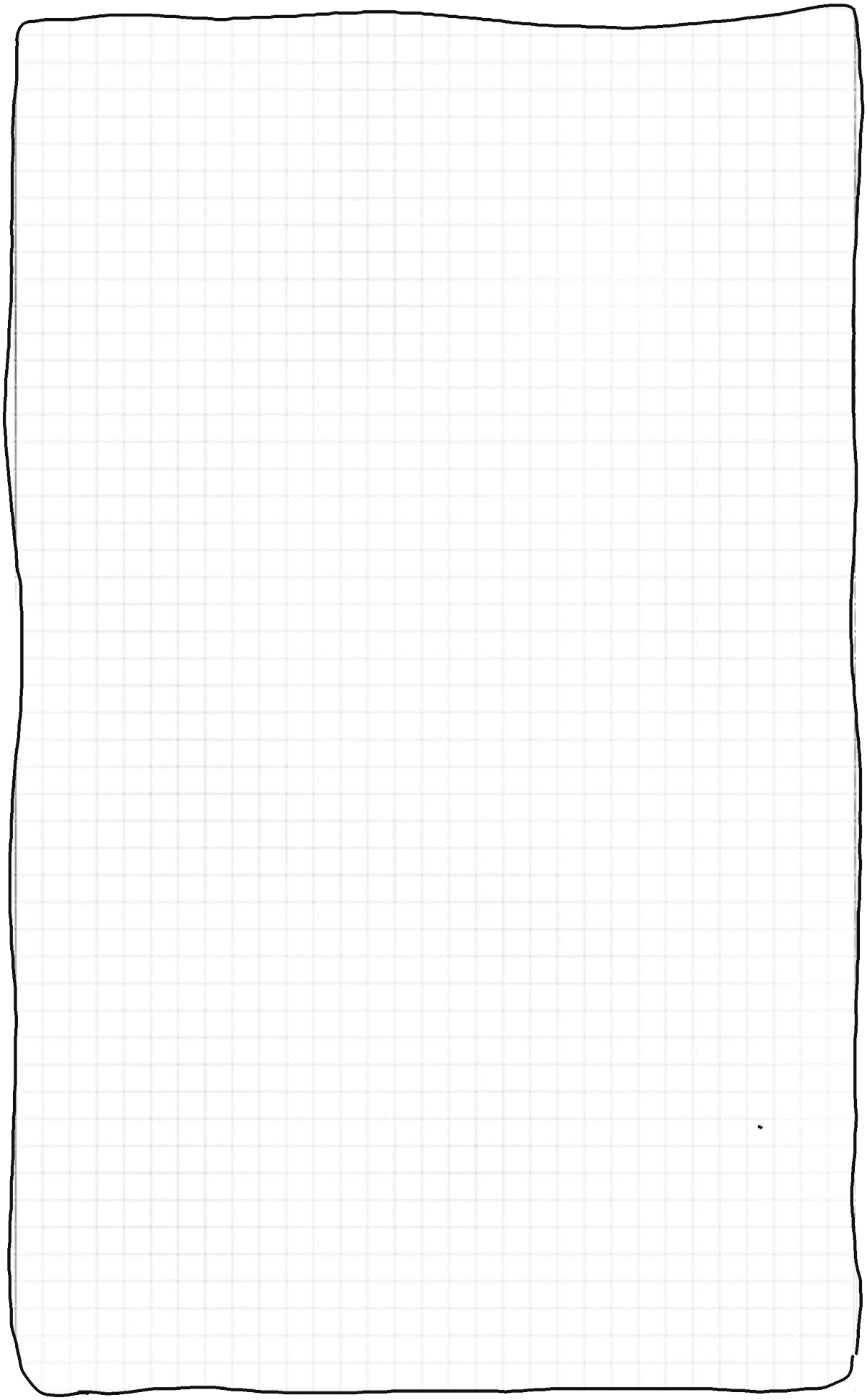
$$f^{\text{concat}}(h_{t-1}^{dec}, h^{enc}) = v_{\text{attn}}^{T} \tanh(w_{\text{attn}}[h_{t-1}^{dec}; h^{enc}])$$

$$f^{\text{general}}(h_{t-1}^{dec}, h^{enc}) = {h_{t-1}^{dec}}^{T} w_{\text{attn}} h^{enc}$$

$$f^{\text{dot}}(h_{t-1}^{dec}, h^{enc}) = {h_{t-1}^{dec}}^{T} h^{enc}$$

M.-T. Luong, H. Pham, Ch.D. Manning,
Effective Approaches to Attention-based Neural Machine Translation,
arXiv:1508.04025 [cs.CL], 2015.

Umsetzung des *multiplicative-Style:*



Beispiel-Implementierung ohne Minibatches:

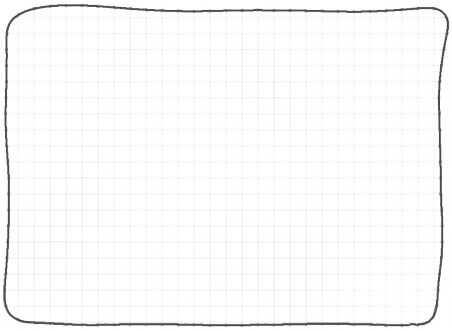
$$f^{\text{general}}(h_{t-1}^{dec}, h^{enc}) = {h_{t-1}^{dec}}^T w_{\text{attn}} h^{enc}$$

```
# general:
#
struct AttnGeneral
    w
    AttnGeneral(n_units) = new(param(n_units, n_units))
end

# simple: only w/o minibatch
#
function (attn::AttnGeneral)(h_t, h_enc)   # h_t is (n_units),
                                           # h_enc is (n_units, n_steps)

    attn_score = permutedims(h_t) * attn.w * h_enc
    α = softmax(attn_score)

    context = h_enc * permutedims(α)
    return context, α
end
```

```
a = AttnGeneral(3)
a(h_t, h_enc)
```

In vereinfachter Einstein-Notation:

$$f^{\text{general}}(h_{t-1}^{dec}, h^{enc}) = {h_{t-1}^{dec}}^T w_{\text{attn}} h^{enc}$$



```
@einsum a[mb,t] := h_t[ud,mb] * w[ud,ue] * h_enc[ue,mb,t]
```

Beispiel-Implementierung mit Minibatches:

$$f^{\text{general}}(h_{t-1}^{dec}, h^{enc}) = {h_{t-1}^{dec}}^T w_{\text{attn}} h^{enc}$$

```
struct AttnGeneral
    projection
    AttnGeneral(n_units) = new(Linear(n_units, n_units, bias=false, actf=identity))
end

function (attn::AttnGeneral)(h_t, h_enc)    # h_t is (n_units,<n_mb>),
                                            # h_enc is (n_units, <n_mb>, n_steps)

    units = size(h_enc)[1]
    steps = size(h_enc)[ndims(h_enc)]
    h_encR = reshape(h_enc, units, :, steps)
    mb = size(h_enc)[2]
    h_tR = reshape(h_t, units, :)

    proj = attn.projection(h_encR)
    attn_score = sum(proj .* h_tR, dims=1)
    α = softmax(attn_score, dims=3)

    c = sum(h_enc .* α, dims=3)

    # remove unneeded dims:
    #
    c = reshape(c, units, mb)
    α = reshape(α, mb, steps)

    return c, α
end
```

11.4 Dot-Product Attention

Beispiel-Implementierung ohne Minibatches:

$$f^{\text{dot}}(h_{t-1}^{dec}, h^{enc}) = {h_{t-1}^{dec}}^T h^{enc}$$

```
# dot:
#
struct AttnDot
end

# simple: only w/o minibatch
#
function (attn::AttnDot)(h_t, h_enc)   # h_t is (n_units),
                                       # h_enc is (n_units, n_steps)

    steps = size(h_enc)[2]

    attn_score = permutedims(h_t) * h_enc
    α = softmax(attn_score)

    context = h_enc * permutedims(α)
    return context, α
end
```

Beispiel-Implementierung mit Minibatches:

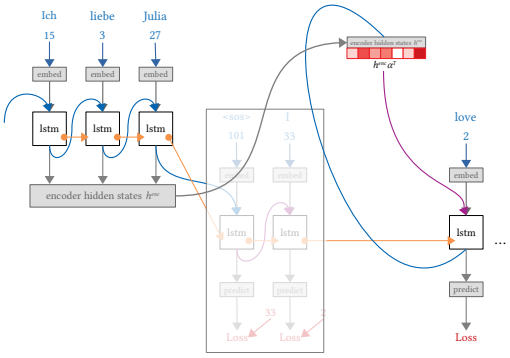
$$f^{\text{dot}}(h_{t-1}^{dec}, h^{enc}) = {h_{t-1}^{dec}}^T h^{enc}$$

```
# dot:
#
struct AttnDot
end

function (attn::AttnDot)(h_t, h_enc)   # h_t is (n_units,<n_mb>),
                                       # h_enc is (n_units, <n_mb>, n_steps)

    # make all 3d:
    #
    h_encR = reshape(h_enc, size(h_enc)[1], :, size(h_enc)[ndims(h_enc)])
    units, mb, steps = size(h_encR)
    h_tR = reshape(h_t, size(h_t)[1], :)

    attn_score = sum(h_encR .* h_tR, dims=1)
    α = softmax(attn_score)

    c = sum(h_enc .* α, dims=3)

    # remove unneeded dims:
    #
    c = reshape(c, units, mb)
    α = reshape(α, mb, steps)

    return c, α
end
```

11.5 Attention für besonere Fälle

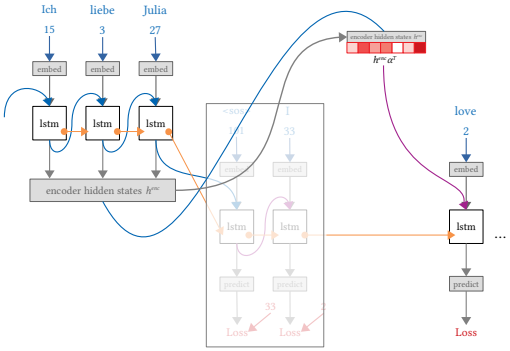
Location-based Attention

$$\alpha^{\text{loc}} = \text{softmax}({h_{t-1}^{dec}}^T w_{\text{attn}})$$

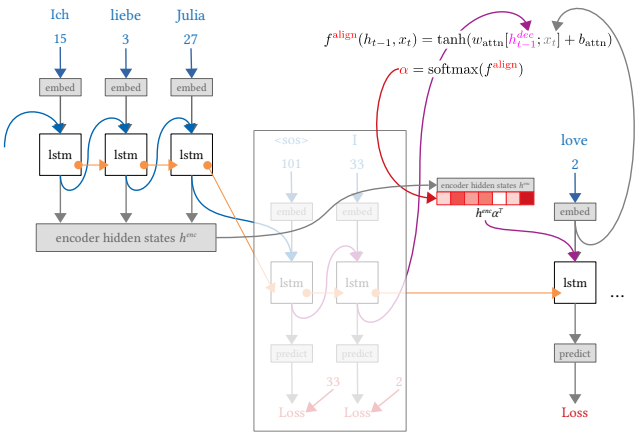


Temporal Attention

$$\alpha^{\text{loc}} = \text{softmax}({h_{t-1}^{dec}}^T w_{\text{attn}})$$



Input Feeding



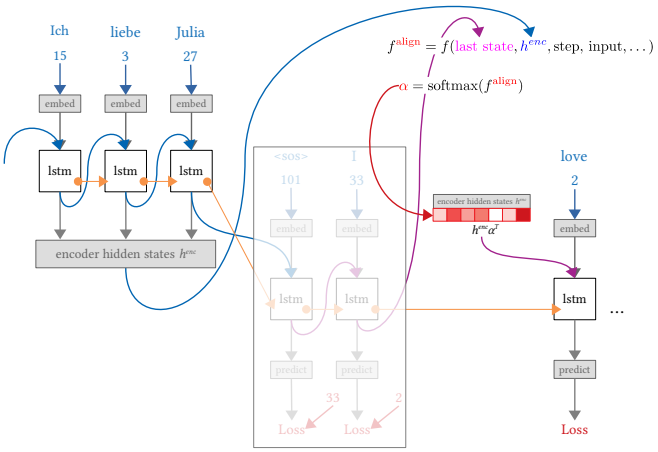
Umsetzung Input Feeding

$$f^{\text{align}}(h_{t-1}, x_t) = \tanh(w_{\text{attn}}[h_{t-1}^{dec}; x_t])$$

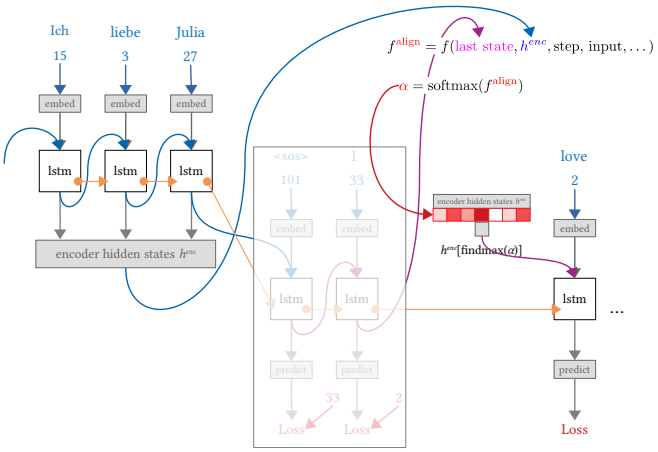
```
# infeed:
#
struct AttnInFeed
    combine
    AttnInFeed(n_units, n_embed, max_seq) = new(Dense(n_units+n_embed, max_seq,
                                                      actf=tanh))
end

# simple: only w/o minibatch
#
function (attn::AttnInFeed)(h_t, h_enc, x)        # h_t is (n_units),
                                                  # h_enc is (n_units, n_steps)
                                                  # x is (n_embed)

    attn_score = attn.combine(vcat(h_t, x))
    α = softmax(attn_score)

    context = h_enc * α
    return context, α
end
```

11.6 Soft vs. hard Attention

Soft:

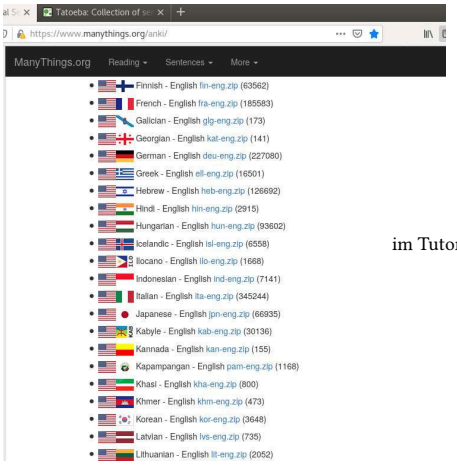


Hard:



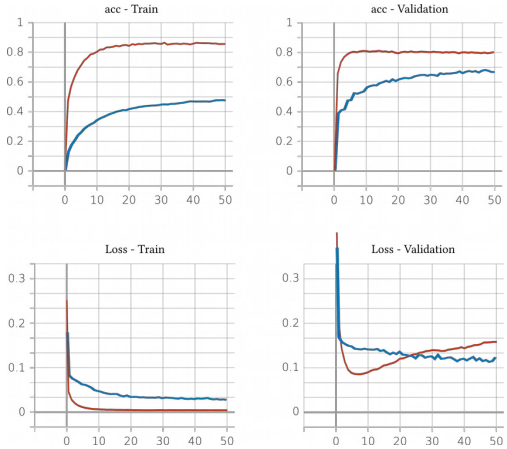
11.7 Beispiel: Machine Translation

Datensatz



im Tutorial!

Erste Trainingsversuche: 50 Epochen



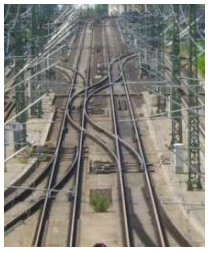
MB: 16

MB: 32
dropout: 0.3
l2: $2e^{-6}$

Ergebnisse:

Ohne Attention:

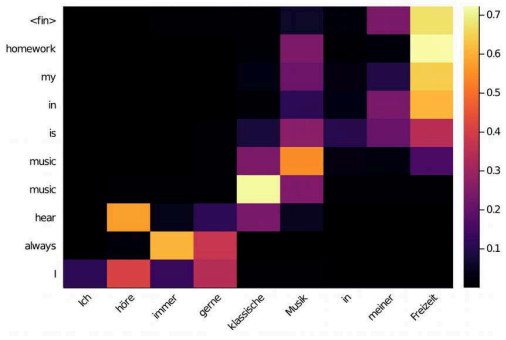
- "Ich liebe Julia" → "I love Julia"
- "Peter liebt Python" → "Peter loves Python"
- "Ich liebe Python" → "I love Julia"



Mit Attention:

- "Ich liebe Julia" → "I love Julia"
- "Peter liebt Python" → "Peter loves Python"
- "Ich liebe Python" → "I love Python"

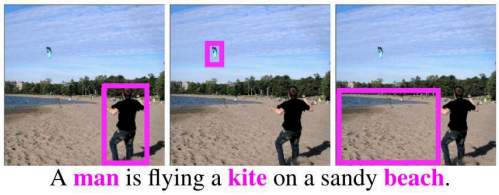
Ergebnisse: Attention

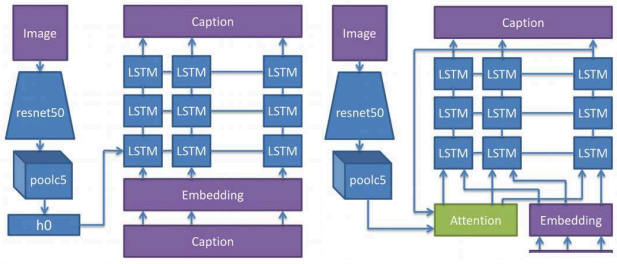



```
Julia> o,α = translate(mdl,
"Ich höre immer gerne klassische Musik in meiner Freizeit")
"I always hear music music is in my homework <fin>"
```

11.8 Attention mit CNNs

Image Captioning





Rister, Blaine and D. Lawson.
Image Captioning with Attention.
cs231n.stanford.edu, (2016).

erfolgreiche Beispiele

Rister, Blaine and D. Lawson.
Image Captioning with Attention.
cs231n.stanford.edu, (2016).

Fehlerhafte Beispiele

A large white bird standing in a forest.

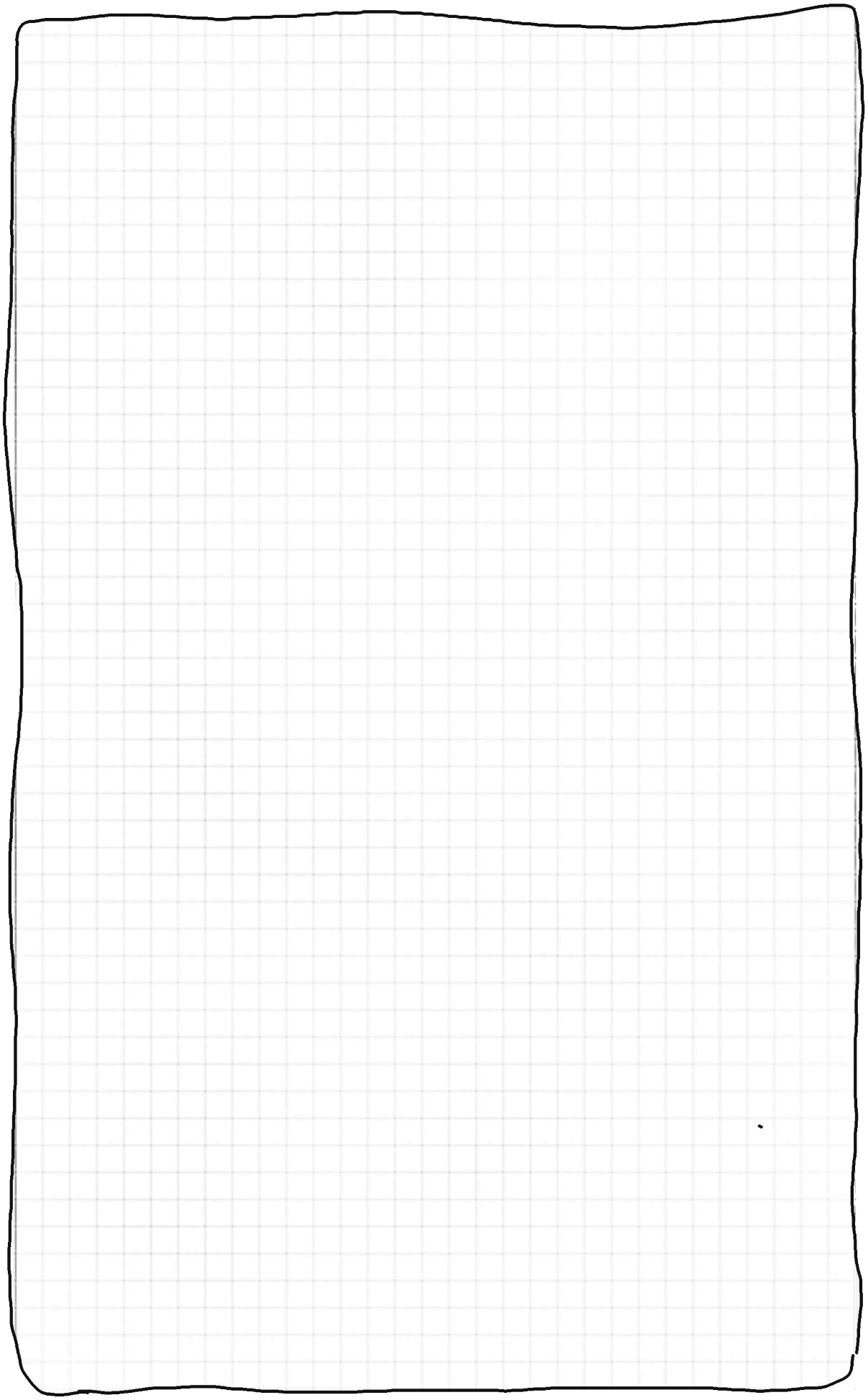
A woman holding a clock in her hand.

Summary

- Ursprünglich für Seq-2-Seq erfunden
- direkt möglich bei Seq-Classifier
- auch möglich bei CNNs (Region der Feature Matrix)

Je nach Anwendung eher

- syntaktisch
- sematisch
- location-based
- temporal

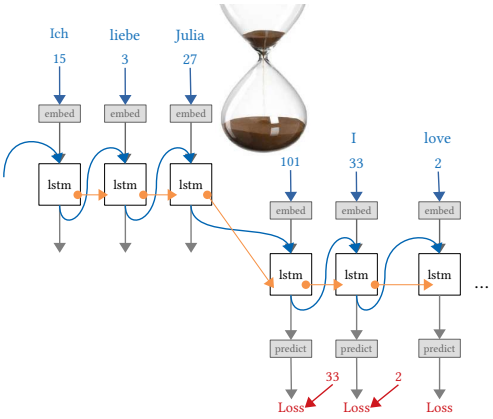


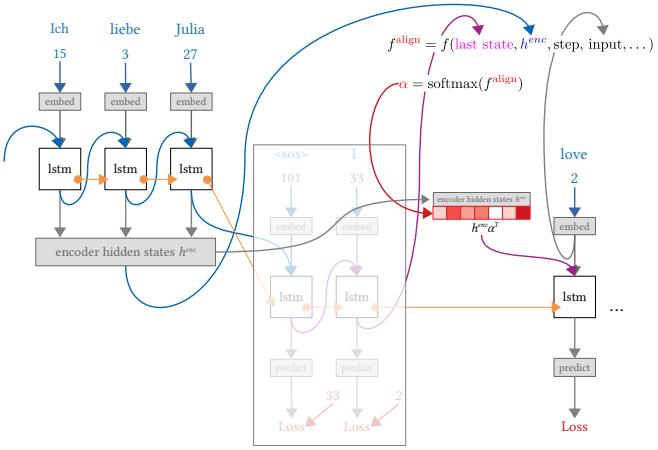
Kapitel 12:

Transformer - Attention is All you Need

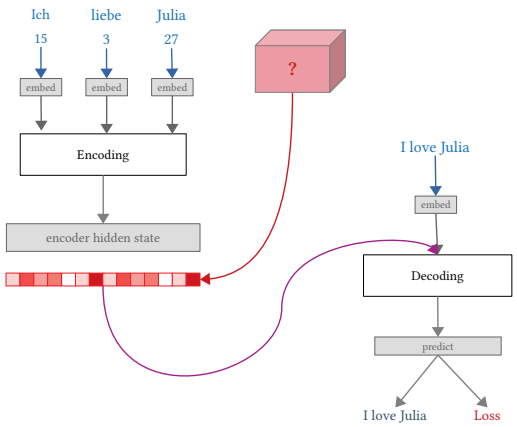
12.1 Attention

Attention im RBB





Attention auch ohne RNN möglich?



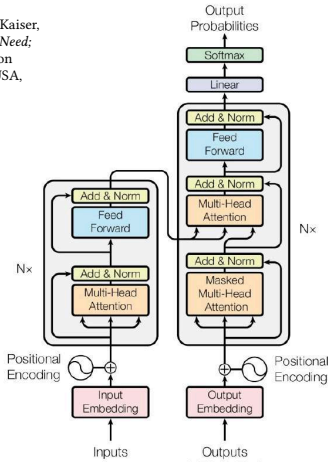
Paper: Attenion is All You Need (2017)

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Google Brain:
Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. *Attention Is All You Need;* 31st Conference on Neural Information Processing SystemLong Beach, CA, USA, 2017.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. *Attention Is All You Need;* 31st Conference on Neural Information Processing SystemLong Beach, CA, USA, 2017.



Embedding und Encoding

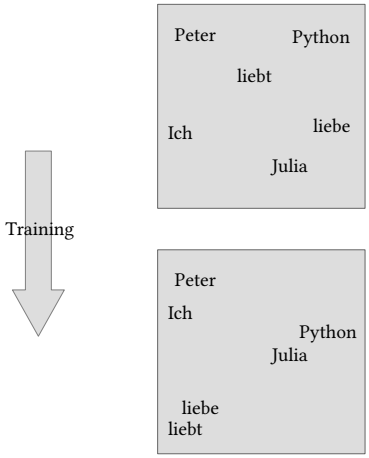
Tokenisation

```
de = ["Ich liebe Julia",
      "Peter liebt Python"]
de_vocab = WordTokenizer(de)
```

```
WordTokenizer
  len → 10
  w2i → Dict{String, Int64} with 10 entries
        "Peter" => 1
        "Ich" => 2
        "liebe" => 3
        "<end>" => 8
        "<pad>" => 9
        "liebt" => 4
        "Julia" => 5
        "Python" => 6
        "<start>" => 7
        "<unknown>" => 10
```

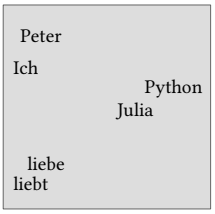
- Nummer pro Wort (besser Token)
- zufällig
- eindeutig

Trainierbares Embedding



aus

- Training
- vortrainiert
- überlegt



Peter	1.5,	10.0
Ich	1.0,	9.0
Python	10.0,	5.5
Julia	8.5,	5.0
liebe	2.0,	2,0
Liebt	1.5,	1,5

```
struct Embed
    w
    Embed(v_siz, embed) = new(Knet.param(embed, v_siz))
end

(l::Embed)(x) = l.w[:,x]
```

Positional Encoding

- Position des Tokens soll im Embedding sichtbar sein!

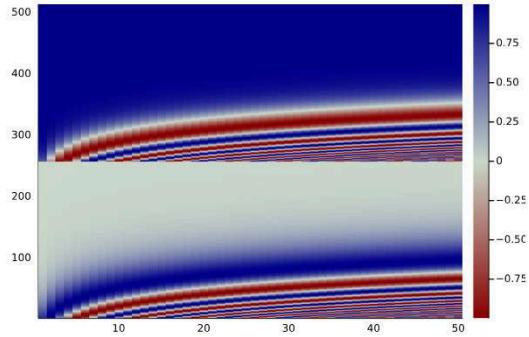
Peter	1.5,	10.0
Ich	1.0,	9.0
Python	10.0,	5.5
Julia	8.5,	5.0
liebe	2.0,	2,0
Liebt	1.5,	1,5

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

```
function positional_encoding(n_embed, n_seq)

    angl = [1/(10000^(2*i/n_embed)) for i in 1:n_embed/2]
    angl = angl * permutedims(1:n_seq)
    pe = vcat(sin.(angl), cos.(angl))
    return pe
end
```

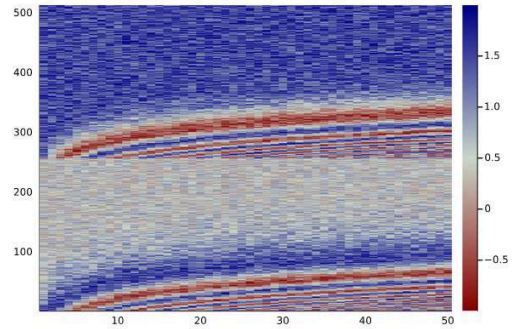
- Position des Tokens soll im Embedding sichtbar sein!

```
n_seq = 50
n_embed = 512
pe = positional_encoding(n_embed, n_seq)
heatmap(pe, c=:redsblues)
```



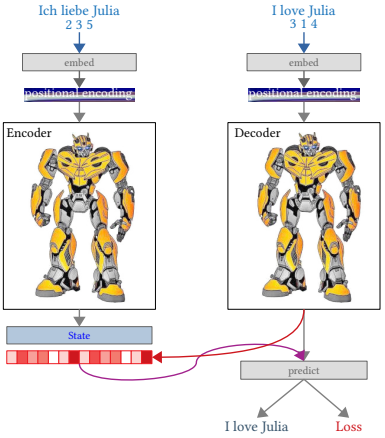
- Position des Tokens soll im Embedding sichtbar sein!

```
n_seq = 50
n_embed = 512
pe = positional_encoding(n_embed, n_seq)
heatmap(rand(n_embed, n_seq).+pe, c=:redsblues)
```

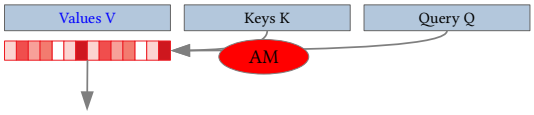


Dot-Product Selfattention

erste Idee des Transfromers

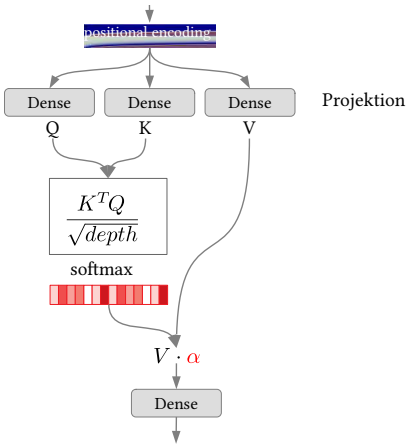


Attention aus K, Q und V



Alles möglich
Einfachste Variante:
Dot-Product-Attention
(Achtung: Matrix-Produkt **nicht** Skalarprodukt!)

Scaled Dot-Product Selfattention



Matrix der Attention-Faktoren

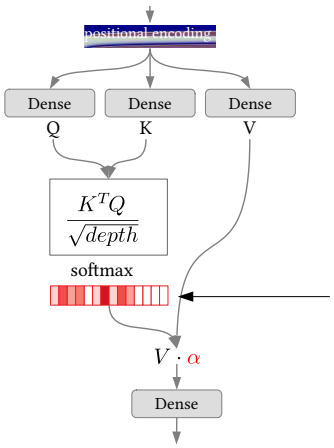
Ich	**Ich**	liebe	**Julia**	pad	pad
liebe	Ich	**liebe**	Julia	pad	pad
Julia	**Ich**	liebe	**Julia**	pad	pad
pad	Ich	liebe	Julia	**pad**	**pad**
pad	Ich	liebe	Julia	**pad**	**pad**

2 Probleme:

- Padding soll ignoriert werden!
- fast die gesamte Attn geht auf die Diagonale!

Maskierung

Attention für maskierte Positionen soll 0.0 sein:



Wie maskieren, dass *nach* softmax() die 0.0 entsteht?

```
using Knet
```

```
x = [0.5 6.0 -1.5 10.5 -5.5]
```

```
1×5 Array{Float64,2}:
 0.5  6.0  -1.5  10.5  -5.5
```

```
softmax(x)
```

```
1×5 Array{Float64,2}:
 4.48988e-5  0.0109864  6.0764e-6  0.988963  1.11293e-7
```

softmax()

```
using Knet
```

```
x = [0.5 6.0 -1.5 10.5 -5.5]
```

```
1×5 Array{Float64,2}:
 0.5  6.0  -1.5  10.5  -5.5
```

```
softmax(x)
```

```
1×5 Array{Float64,2}:
 4.48988e-5  0.0109864  6.0764e-6  0.988963  1.11293e-7
```

```
softmax(x) .* [1 1 1 0 0]
```

```
1×5 Array{Float64,2}:
 4.48988e-5  0.0109864  6.0764e-6  0.0  0.0
```

hinterher maskieren

```
using Knet
```

```
x = [0.5 6.0 -1.5 10.5 -5.5]
```

```
1×5 Array{Float64,2}:
 0.5  6.0  -1.5  10.5  -5.5
```

```
softmax(x)
```

```
1×5 Array{Float64,2}:
 4.48988e-5  0.0109864  6.0764e-6  0.988963  1.11293e-7
```

```
softmax(x) .* [1 1 1 0 0]
```

```
1×5 Array{Float64,2}:
 4.48988e-5  0.0109864  6.0764e-6  0.0  0.0
```

```
softmax(x .* [1 1 1 0 0])
```

```
1×5 Array{Float64,2}:
 0.00404792  0.990494  0.000547827  0.00245519  0.00245519
```

vorher maskieren

```
using Knet
```

```
x = [0.5 6.0 -1.5 10.5 -5.5]
```

```
1×5 Array{Float64,2}:
 0.5  6.0  -1.5  10.5  -5.5
```

```
softmax(x)
```

```
1×5 Array{Float64,2}:
 4.48988e-5  0.0109864  6.0764e-6  0.988963  1.11293e-7
```

```
softmax(x) .* [1 1 1 0 0]
```

```
1×5 Array{Float64,2}:
 4.48988e-5  0.0109864  6.0764e-6  0.0  0.0
```

```
softmax(x .* [1 1 1 0 0])
```

```
1×5 Array{Float64,2}:
 0.00404792  0.990494  0.000547827  0.00245519  0.00245519
```

```
M = -1e9
softmax(x .+ [0 0 0 M M])
```

```
1×5 Array{Float64,2}:
 0.0040679  0.995382  0.00055053  0.0  0.0
```

mit -1 000 000 000 maskieren

```
x = [0.5 6.0 -1.5 10.5 -5.5]
mask = [0 0 0 1 1]

masked_softmax(x, mask) = softmax(x .+ -1e9 .* mask)

masked_softmax(x, mask)
```

```
1×5 Array{Float64,2}:
 0.0040679  0.995382  0.00055053  0.0  0.0
```

mit vorbereiteter Maske

Für die Minibatch

Ich liebe Julia
Peter mag Python viel lieber

2	4		2	4
3	8		3	8
5	6		5	6
	9		**0**	9
	7		**0**	7

Padding:

```
function padding_mask(x; pad=0)
    return x .== pad
end

x = [2 4
     3 8
     5 6
     0 9
     0 7]

padding_mask(x)
```

```
5×2 BitArray{2}:
 0  0
 0  0
 0  0
 1  0
 1  0
```

Ich liebe Julia
Peter mag Python viel lieber

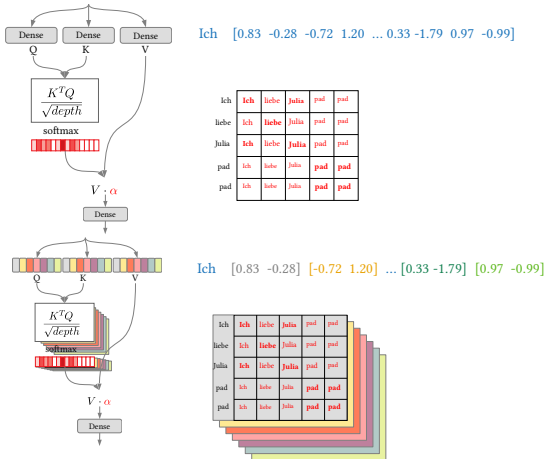
```
function padding_mask_mha(x; pad=0)
    return reshape(x .== pad, size(x)[1],1,1,size(x)[2])
end

x = [2 4
     3 8
     5 6
     0 9
     0 7]
padding_mask_mha(x)
```

```
5×1×1×2 BitArray{4}:
[:, :, 1, 1] =
 0
 0
 0
 1
 1

[:, :, 1, 2] =
 0
 0
 0
 0
 0
```

Implementierung der Attention

```
function dot_prod_attn(q, k, v; mask=nothing)

    score = bmm(k, q, transA=true) ./ Float32(sqrt(size(q)[1]))  # [s_q, s_k, ..., mb]
    if mask != nothing
        score = score .+ mask * Float32(-1e9)
    end

    α = softmax(score, dims=1)
    c = bmm(v, α)
    return c, α
end
```

Multi-Headed Selfattention




```
function separate_heads(x, n)
    depth, seq, mb = size(x)
    mh_depth = depth ÷ n
    x = reshape(x, mh_depth, n, :, mb)
    return permutedims(x, (1,3,2,4))
end

function merge_heads(x)
    mh_depth, seq, n, mb = size(x)
    depth = mh_depth * n
    x = permutedims(x, (1,3,2,4))
    return reshape(x, depth, :, mb)
end
```

1) Embedding-Tiefe teilen in n Heads und eine ebene pro Head einfügen (zusätzliche Dimension)
2) Sequenzlänge zurück in 2. Dim

⟶ head-depth * seq-len * n-heads * n-minibatch

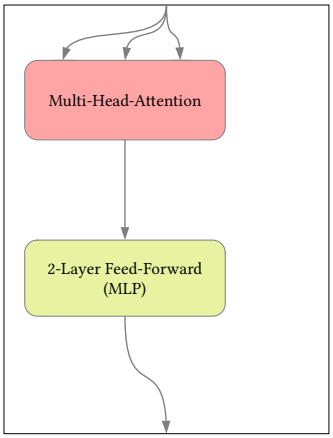
α ⟶ seq-len1 * seq_len2 * n-heads * n-minibath

```
function(mha::MultiHeadAttn)(q, k, v; mask=nothing)

    q = mha.dense_q(q)                          # [depth, n_seq, n_mb]
    k = mha.dense_q(k)
    v = mha.dense_q(v)

    q = separate_heads(q, mha.n_heads)          # [depth/n, n_seq, n_heads, n_mb]
    k = separate_heads(k, mha.n_heads)
    v = separate_heads(v, mha.n_heads)

    c, α = dot_prod_attn(q, k, v, mask=mask)    # c: [depth/n, n_seq, n_heads, n_mb]
                                                # α: [n_seq, n_seq, n_heads, n_mb]
    c = merge_heads(c)                          # [depth, n_seq_ n_mb]
    return mha.dense_out(c), α
end
```

1) Projektionen von Q, K, V berechnen
2) Embedding-Tiefe teilen in n Heads
3) Attention
4) Heads zusammenfassen
5) Projektion zurückliefern

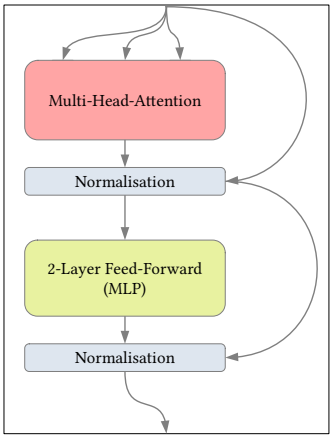
12.2 Der Encoder

... unfertig



n = 8 - 24

mit Residual-Connections

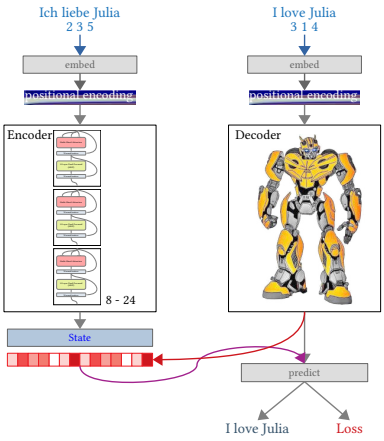


Der Encoder lernt die Quellsprache.

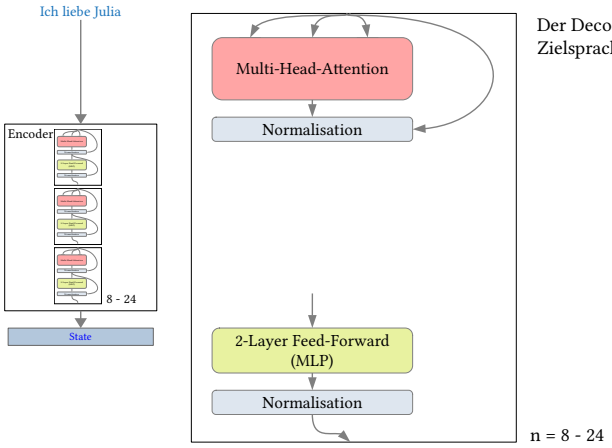
n = 8 - 24

12.3 Der Decoder

Encoder im Transformer

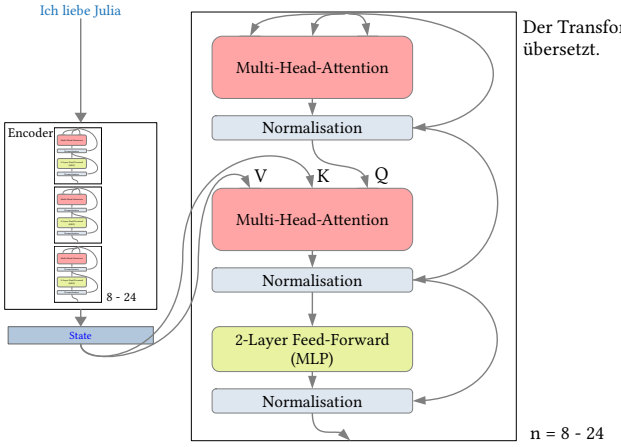


Selfattention im Decoder



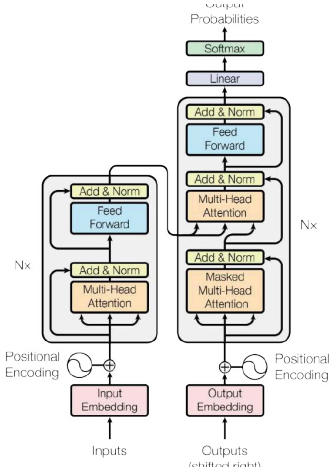
Der Decoder lernt die Zielsprache.

fertig!

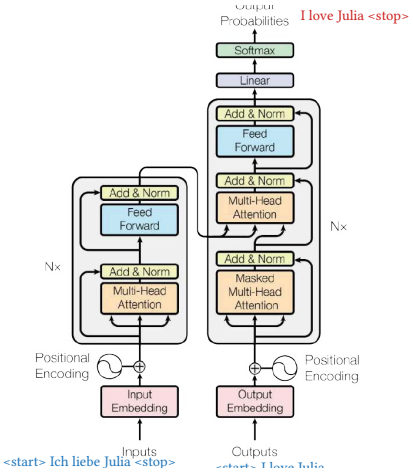


Der Transformer übersetzt.

Transformer in der Google-Darstellung



Maskierte Selfattention mit Peek-ahead Mask



Problem:

- Im RNN wird immer ein Token präsentiert → Vorhersage nötig, keine Schummelei möglich.
- Im Transformer wird die ganze Sequenz präsentiert → keine Vorhersage nötig! Nur shift:

<start> Ich liebe Julia → I love Julia <end>

	<start>	I	love	Julia	pad
<start>	<start>	<start>	<start>	<start>	<start>
I	I	**I**	I	**I**	I
love	love	love	love	love	love
Julia	Julia	Julia	Julia	**Julia**	Julia
pad	pad	pad	pad	pad	**pad**

	<start>	I	love	Julia	pad
<start>	<start>	<start>	<start>	<start>	<start>
I	mask	**I**	I	**I**	I
love	mask	mask	love	love	love
Julia	mask	mask	mask	**Julia**	Julia
pad	mask	mask	mask	mask	**pad**

Look-ahead Mask und Padding Mask:

	<start>	I	love	Julia	pad
<start>	<start>	<start>	<start>	<start>	<start>
I	mask	**I**	I	**I**	I
love	mask	mask	love	love	love
Julia	mask	mask	mask	**Julia**	Julia
pad	mask	mask	mask	mask	mask

```
function peek_ahead_mask(x)
    n_seq = size(x)[1]

    return 1 .- UpperTriangular(ones(n_seq, n_seq))
end
```

- ... wenn möglich mit *triangular()*-Funktion

```
UpperTriangular(ones(5,5))
```

```
5×5 UpperTriangular{Float64,Array{Float64,2}}:
 1.0  1.0  1.0  1.0  1.0
  ·   1.0  1.0  1.0  1.0
  ·    ·   1.0  1.0  1.0
  ·    ·    ·   1.0  1.0
  ·    ·    ·    ·   1.0
```

```
function padding_mask(x; pad=0)

    return reshape(x .== pad, size(x)[1],1,1,size(x)[2])
end

function peek_ahead_mask(x)
    n_seq = size(x)[1]

    return 1 .- UpperTriangular(ones(n_seq, n_seq))
end

x = [2 4
     3 8
     5 6
     0 9
     0 7]
```

```
5×2 Array{Int64,2}:
 2  4
 3  8
 5  6
 0  9
 0  7
```

```
max.(padding_mask(x) , peek_ahead_mask(x))
```

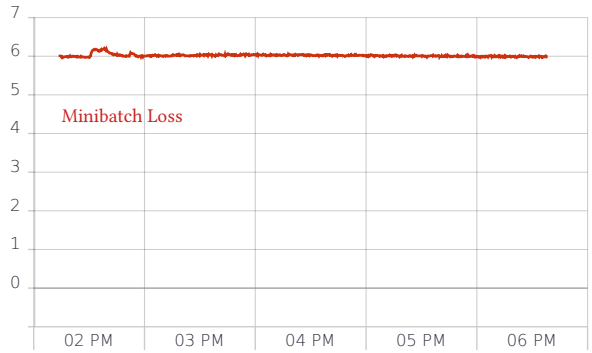
```
5×5×1×2 Array{Float64,4}:
[:, :, 1, 1] =
 0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0
 1.0  1.0  0.0  0.0  0.0
 1.0  1.0  1.0  1.0  1.0
 1.0  1.0  1.0  1.0  1.0

[:, :, 1, 2] =
 0.0  0.0  0.0  0.0  0.0
 1.0  0.0  0.0  0.0  0.0
 1.0  1.0  0.0  0.0  0.0
 1.0  1.0  1.0  0.0  0.0
 1.0  1.0  1.0  1.0  0.0
```

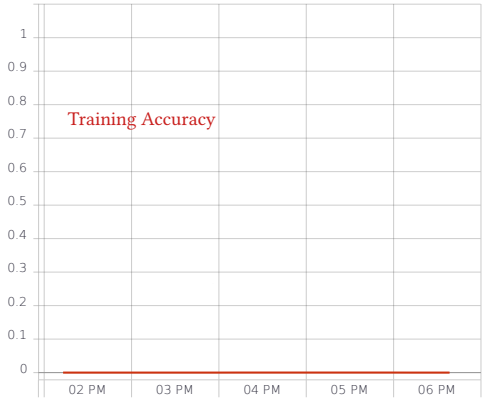
Dims sind:
(wie α -Tensor,
bzw. broadcastable mit α)

[seq-len, seq-len, heads, minibatch]

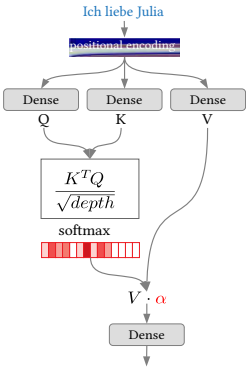
12.4 Training des Transformers

Training mit $\mu = 0.001$



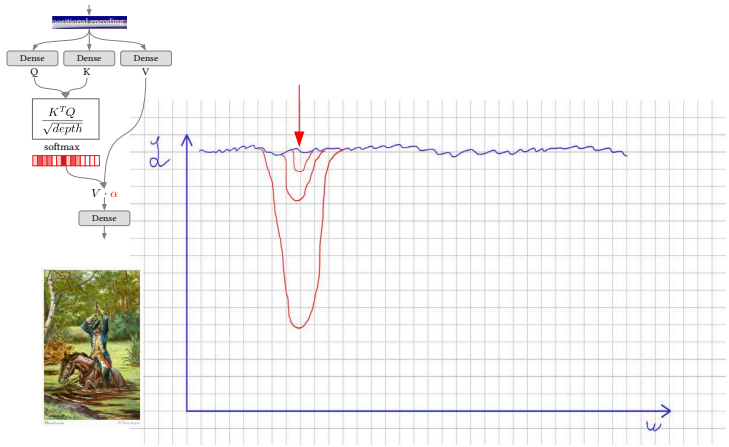


Warum sollte das auch funktionieren?

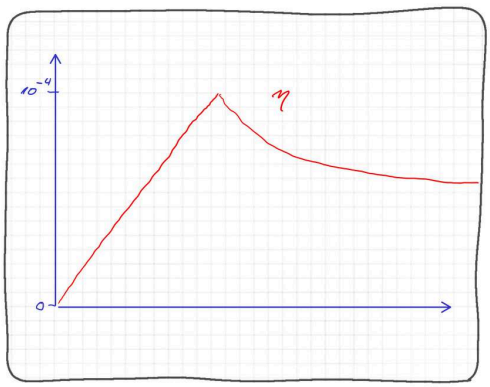


Das Ergebnis soll den Sinn des Satzes repräsentieren?

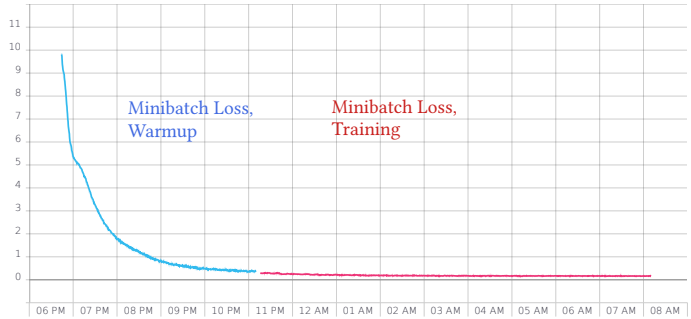
Winzige Lernrate erzeugt eine loake *Delle* in der Fehlerfläche...

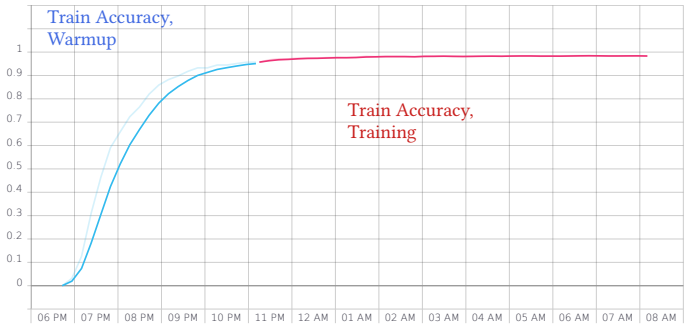


Lernratenstrategie

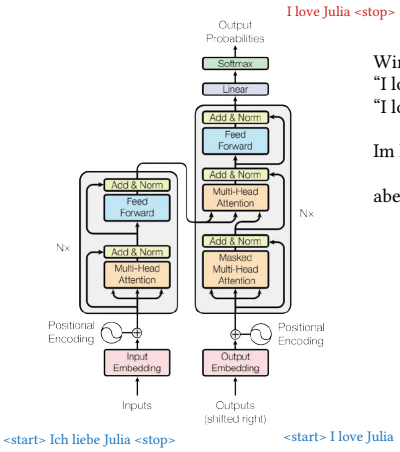



```
ayn = AllYouNeed(n_layers=6, depth=256, n_heads=8,
                 x_vocab=d_vocab, y_vocab=e_vocab, drop_rate=0.1)

ayn = tb_train!(ayn, Adam, mbs, epochs=20,
                lr=1e-9, lr_decay=2e-4, lrd_freq=20, lrd_linear=true,
                beta2=0.98, eps=1e-9,
                tb_name="tatoeba_fixed_WARMUP",
                acc_fun=train_acc, eval_size=0.01, eval_freq=1
                )
Knet.gc()

ayn = tb_train!(ayn, Adam, mbs, epochs=60,
                lr=2e-4, lr_decay=1e-5, lrd_freq=0.5, lrd_linear=false,
                beta2=0.98, eps=1e-9,
                tb_name="tatoeba_fixed_TRAIN",
                acc_fun=train_acc, eval_size=0.01, eval_freq=1
                )
```


Minibatch Loss,
Warmup
Minibatch Loss,
Training
11
10
9
8
7
6
5
4
3
2
1
0
06 PM
07 PM
08 PM
09 PM
10 PM
11 PM
12 AM
01 AM
02 AM
03 AM
04 AM
05 AM
06 AM
07 AM
08 AM


Train Accuracy,
Warmup
Train Accuracy,
Training
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
06 PM
07 PM
08 PM
09 PM
10 PM
11 PM
12 AM
01 AM
02 AM
03 AM
04 AM
05 AM
06 AM
07 AM
08 AM

12.5 Prediction

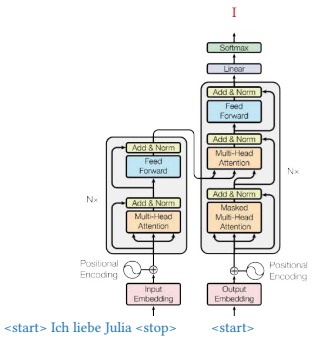


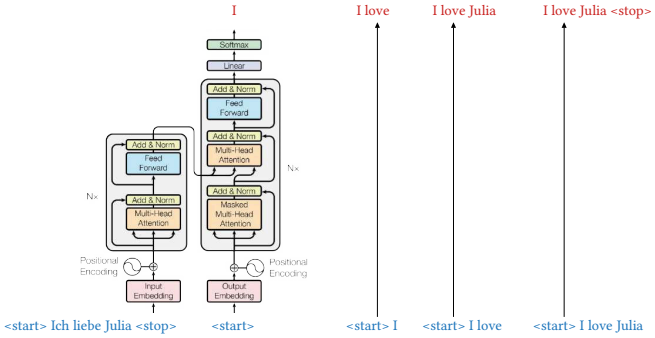
Wir können:
"I love Julia" in
"I love Julia" transformieren,

Im Kontext von "Ich liebe Julia",

aber ...

Loop





- Encoder muss nur einmal berechnet werden
- 2 Möglichkeiten, die Sequenz zu bekommen:
  - immer letztes Zeichen anhängen
  - immer ganze transformierte Sequenz berichten.

Optimierung

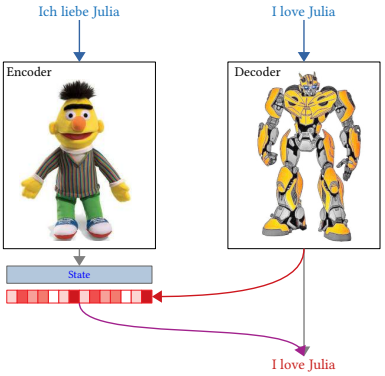
- Zunächst: langsam
  - Groß, viele Parameter
  - Kleine Lernrate
  - Warmup
- Aber:
  - Nur Matrix-Multiplikationen → perfekt auf GPUs
  - Training langsam, Prediction optimierbar!

Große Netze lernen schneller als kleine!?!

- Relativ groß starten
- Minibatch size matters!
- Time pro Epoche vs. Wall Time
- dann optimieren:
  - quantisation: Rechengenauigkeit verringern
  - prune: ungenutzte Parameter entfernen


mit ein paar guten Ideen: möglich!

12.6 BERT: Bidirectional Encoder Representations for Transformers



Masked Language Modelling (MLM)

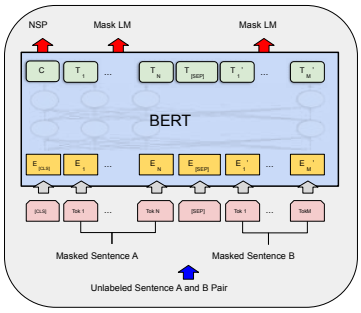
- 15% der Tokens maskiert
- Loss nur für die Predictions
- Tricks:
  von den 15% maskierten Tokens werden maskiert
  - 80% mit [MASK]
    (sonst würden unmaskierte Worte nicht gelernt)
  - 10% mit einem zufälligen Wort
    (sonst könnten die korrekten Worte leicht auswendig gelernt werden – ohne Kontext)
  - 10% mit dem korrekten Wort
    (sonst würde das Modell lernen, dass die Prediction niemals richtig ist)

→ BERT lernt die Sprache!

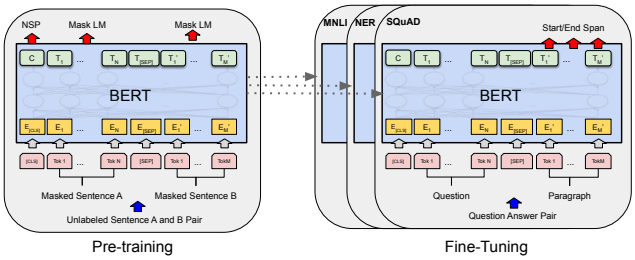
Next Sentence Prediction (NSP)

- Es wird einfach trainiert, ob der 2 Satz logisch korrekt auf den ersten folgt (binäre Klassifizierung).

→ BERT lernt Dialoge!



Pre-training



Pre-training Fine-Tuning

MNLI: Multi-Genre Natural Language Inference Corpus (Williams et al., 2018): crowd- sourced collection of sentence pairs with textual entailment annotations.

NER: Language-Independent Named Entity Recognition; Named entities are phrases that contain the names of persons, organizations, locations, times and quantities. https://www.clips.uantwerpen.be/conll2003/ner/

SQuAD: Stanford Question Answering Dataset: reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable. https://rajpurkar.github.io/SQuAD-explorer/

Pretrained BERT

https://github.com/google-research/bert

The links to the models are here (right-click, 'Save link as...' on the name):

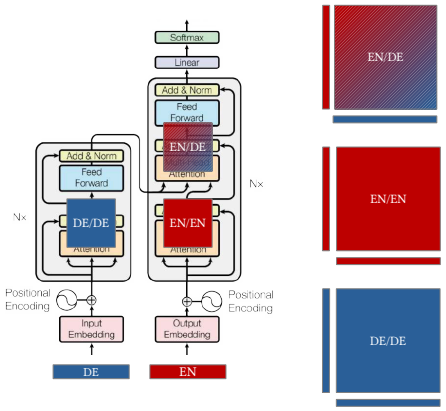
- `BERT-Large, Uncased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Large, Cased (Whole Word Masking)` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Uncased` : 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Large, Uncased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Cased` : 12-layer, 768-hidden, 12-heads , 110M parameters
- `BERT-Large, Cased` : 24-layer, 1024-hidden, 16-heads, 340M parameters
- `BERT-Base, Multilingual Cased (New, recommended)` : 104 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Multilingual Uncased (Orig, not recommended)` **(Not recommended, use** `Multilingual Cased` **instead)**: 102 languages, 12-layer, 768-hidden, 12-heads, 110M parameters
- `BERT-Base, Chinese` : Chinese Simplified and Traditional, 12-layer, 768-hidden, 12-heads, 110M parameters

Each .zip file contains three items:

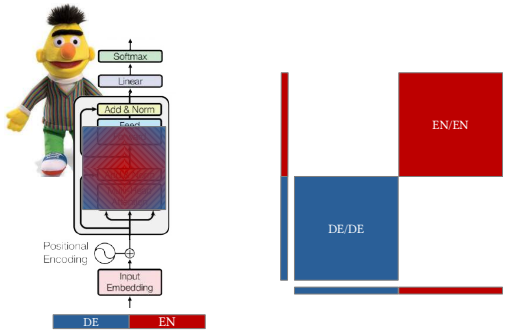
- A TensorFlow checkpoint (`bert_model.ckpt`) containing the pre-trained weights (which is actually 3 files).
- A vocab file (`vocab.txt`) to map WordPiece to word id.
- A config file (`bert_config.json`) which specifies the hyperparameters of the model.

Cross-lingual BERT

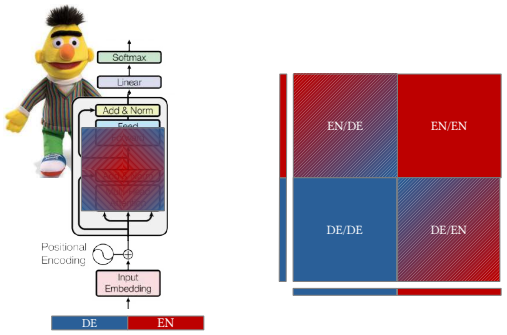
Vanilla Transformer: 3 x Attentionmatrix pro Layer

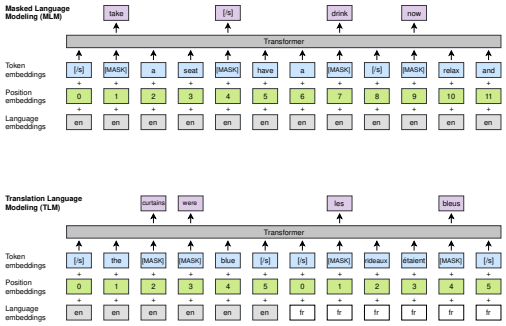


BERT: 1 x Attentionmatrix pro Layer



BERT: 1 x Attentionmatrix pro Layer

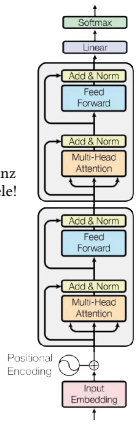




G. Lample, A. Conneau, *Cross-lingual Language Model Pretraining*, 33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

ALBERT

ALBERT: A Light BERT



Ganz viele!

Fragen:

- Warum soll die Tiefe immer gleich der Embedding-Tiefe sein?
  - Vielleicht größer? (Regularisieruung des Embeddings)
  - Entkoppeln!
  - Hidden Size >> Embedding
- Warum sollte die Attention in den Layers unterschiedlich sein?
  - Parameter sharing!
  - Teilweise oder alle über alle Layer!
  - Zusätzliche Layer ⟶ keine zusätzlichen Parameter!

Model		Parameters	Layers	Hidden	Embedding	Parameter-sharing
BERT	base	108M	12	768	768	False
	large	334M	24	1024	1024	False
ALBERT	base	12M	12	768	128	True
	large	18M	24	1024	128	True
	xlarge	60M	24	2048	128	True
	xxlarge	235M	12	4096	128	True

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma and R. Soricut,
ALBERT: A Lite BERT for Self-supervised Learning of Language Representations,
(2020) arXiv:1909.11942 [cs.CL]

Model	E	Parameters	SQuAD1.1	SQuAD2.0	MNLI	SST-2	RACE	Avg
ALBERT base not-shared	64	87M	89.9/82.9	80.1/77.8	82.9	91.5	66.7	81.3
	128	89M	89.9/82.8	80.3/77.3	83.7	91.5	67.9	81.7
	256	93M	90.2/83.2	80.3/77.4	84.1	91.9	67.3	81.8
	768	108M	90.4/83.2	80.4/77.6	84.5	92.8	68.2	82.3
ALBERT base all-shared	64	10M	88.7/81.4	77.5/74.8	80.8	89.4	63.5	79.0
	128	12M	89.3/82.3	80.0/77.1	81.6	90.3	64.0	80.1
	256	16M	88.8/81.5	79.1/76.3	81.5	90.3	63.4	79.6
	768	31M	88.6/81.5	79.2/76.6	82.0	90.6	63.3	79.8

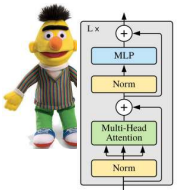
Table 3: The effect of vocabulary embedding size on the performance of ALBERT-base.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma and R. Soricut,
ALBERT: A Lite BERT for Self-supervised Learning of Language Representations,
(2020) arXiv:1909.11942 [cs.CL]

12.7 Transformer für Computer Vision

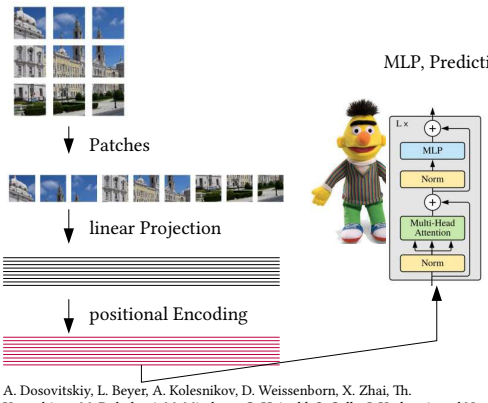
An image is worth 16x16 words

Alexey Dosovitskiy, Intel Labs München



A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, Th. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,* arXiv:2010.11929 [cs.CV]

Alexey Dosovitskiy, Intel Labs München



A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, Th. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,* arXiv:2010.11929 [cs.CV]

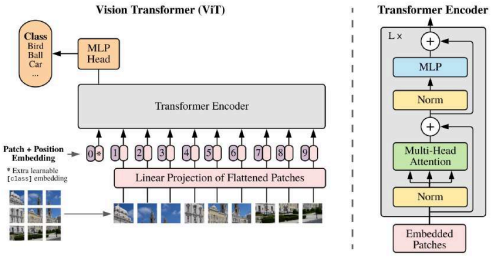
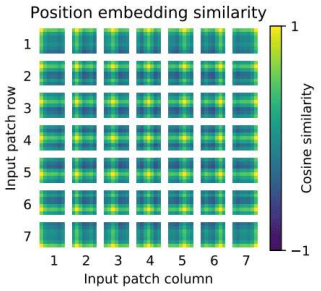
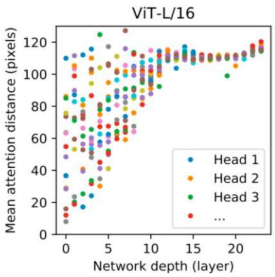


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

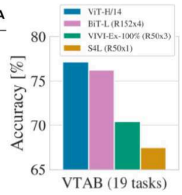
A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, Th. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale,* arXiv:2010.11929 [cs.CV]





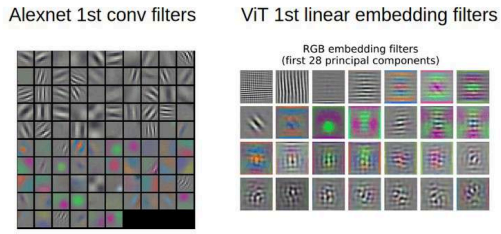
Besser als die CNNs

	ViT-H	Previous SOTA
ImageNet	88.55	88.5
ImageNet-ReaL	90.72	90.55
Cifar-10	99.50	99.37
Cifar-100	94.55	93.51
Pets	97.56	96.62
Flowers	99.68	99.63



Vision Transformer matches or outperforms state-of-the-art CNNs on popular benchmarks. **Left:** Popular image classification tasks (ImageNet, including new validation labels ReaL, and CIFAR, Pets, and Flowers). **Right:** Average across 19 tasks in the VTAB classification suite.

Vergleichbare Filter!



Die Attention zeigt, wie die Entscheidung zustandekommt

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

- Trainiert ab 14 M Trainingsbilder

Fazit:

- Langes Training
- Viele Trainingdaten
- Super einfach
- Self-supervised

- Einfach (Linear und Dense)
- Groß (Switch-Transformer: >>10^9 Params)
- self-supervised (statt plump supervised)