

# Neuronale Netze und Deep Learning

## Geo.BigData(Science)

Dr. Joachim Steinwendner



# Geschichte



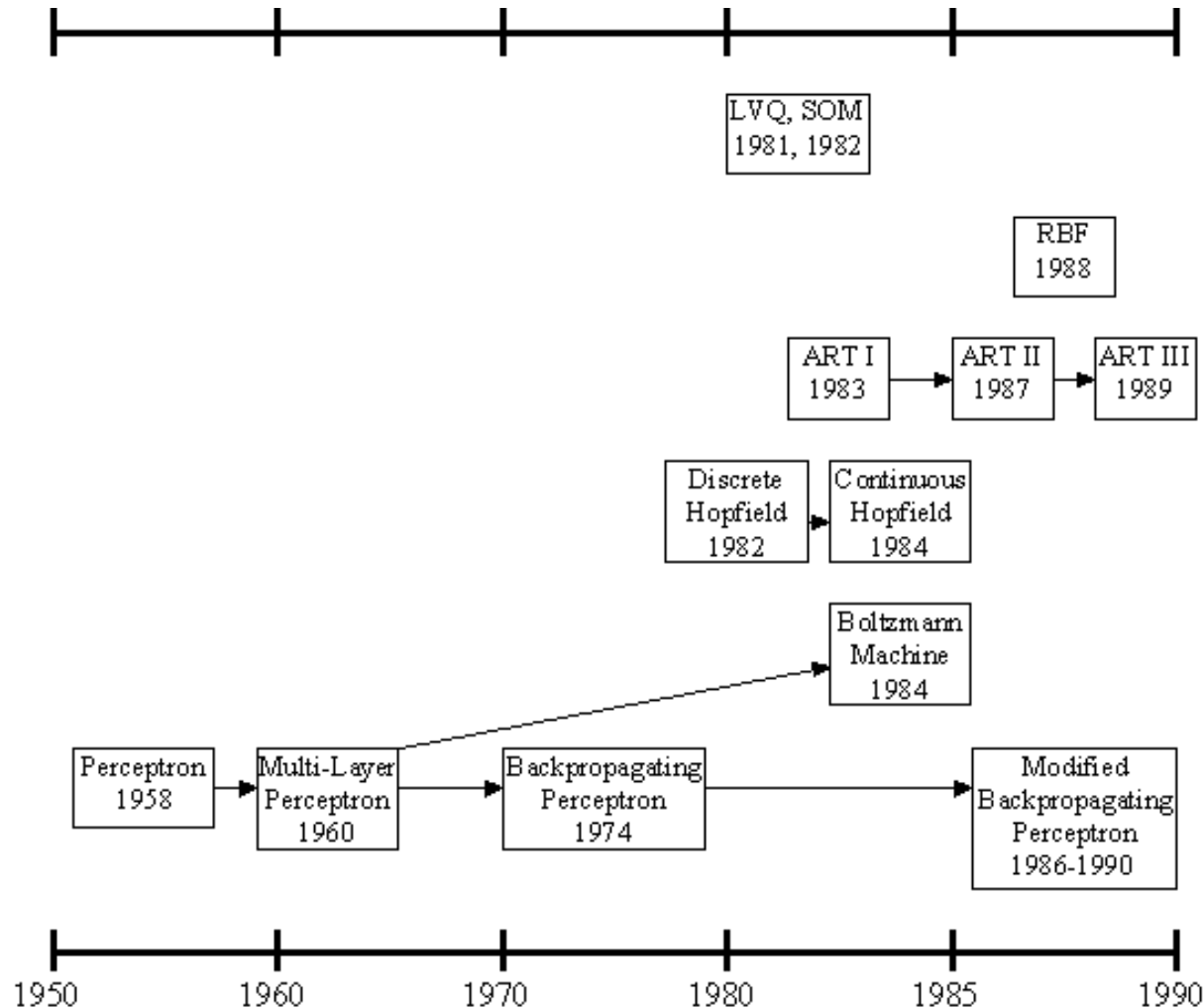
John v. Neumann   Donald O. Hebb   Marvin Minsky   Bernard Widrow   Seymour Papert   Teuvo Kohonen   John Hopfield



Geoffrey Hinton



# Geschichte und Theorie – die Anfänge



**Bereits 1943** beschreiben **WARREN McCULLOCH und WALTER PITTS** eine Art neurologischer Netzwerke, bauen Schwellwertschalter durch Neurone nach und zeigen, dass selbst einfache Netze dieser Art praktisch jede logische oder auch arithmetische Funktion berechnen können [MP43]. Weiter entstehen erste Computervorläufer („**Elektronengehirne**“), u.a. unterstützt von **KONRAD ZUSE**, der es leid war, ballistische Bahnen per Hand zu berechnen.

**1947** nennen **WALTER PITTS und WARREN McCULLOCH** ein praktisches Anwendungsgebiet (in ihrer Arbeit von 1943 wurde noch kein solches genannt), nämlich die Erkennung räumlicher Muster durch Neuronale Netze [PM47].

**1949:** **DONALD O. HEBB** formuliert die klassische *Hebb'sche Lernregel* [Heb49], welche in ihrer allgemeineren Form die Basis fast aller neuronalen Lernverfahren darstellt. Sie besagt, dass die Verbindung zwischen zwei Neuronen verstärkt wird, wenn beide Neurone *gleichzeitig aktiv* sind – die Verstärkung ist also proportional zum Produkt beider Aktivitäten. Hebb konnte diese Regel zwar postulieren, jedoch in Ermangelung neurologischer Forschung nicht verifizieren.

**1950** vertritt Neuropsychologe **KARL LASHLEY** die These, dass die Informationsspeicherung im Gehirn *verteilt* realisiert wird. Begründet wird seine These an Versuchen mit Ratten, bei denen nur der Umfang und nicht der Ort zerstörten Nervengewebes ihre Leistung beeinflusst, aus einem Labyrinth zu finden.



# Geschichte und Theorie – die Blütezeit

- 1951** entwickelt MARVIN MINSKY für seine Dissertation den Neurocomputer *Snark*, der seine Gewichte<sup>9</sup> bereits automatisch justieren kann, aber nie praktisch eingesetzt wird – da er zwar fleißig rechnet, jedoch niemand so genau weiss, was.
- 1956** treffen sich auf dem *Dartmouth Summer Research Project* renommierte Wissenschaftler und aufstrebende Studierende und diskutieren, salopp gesagt, wie man ein Gehirn nachbilden kann – Unterschiede zwischen Top-Down- und Bottom-Up-Forschung bilden sich heraus. Während die frühen Anhänger der *Artificial Intelligence* Fähigkeiten durch Software *nachbilden* möchten, haben die Anhänger der Neuronalen Netze im Sinn, Systemverhalten durch Nachbildung kleinster Systemteile, der Neurone, zu erreichen.
- 1957 - 1958** entwickeln FRANK ROSENBLATT, CHARLES WIGHTMAN und ihre Mitarbeiter am MIT den ersten erfolgreichen Neurocomputer, das *Mark I Perceptron*, welches mit einem  $20 \times 20$  Pixel großen Bildsensor einfache Ziffern erkennen kann und 512 motorbetriebene Potentiometer besitzt - pro variablem Gewicht eins.
- 1959** beschreibt FRANK ROSENBLATT verschiedene Varianten des Perceptrons, formuliert und beweist sein *Perceptron-Konvergenz-Theorem*. Er beschreibt an der Retina orientierte Neuronenschichten, Schwellwertschalter und eine Lernregel, welche die Verbindungsgewichte justiert.
- 1960** stellen BERNARD WIDROW und MARCIAN E. HOFF das *ADALINE (ADaptive LInear NEuron)* vor [WH60], ein schnell und genau lernendes adaptives System, das wohl das erste verbreitet kommerziell eingesetzte Neuronale Netz darstellte: Es war praktisch in jedem Analogtelefon zur Echtzeit-Echofilterung zu finden und lernte mit der *Widrow-Hoff-Lernregel* bzw. *Deltaregel*. Hoff, ein Mitbegründer von Intel, war zu diesem Zeitpunkt Doktorand von Widrow, der seinerseits einer der Erfinder der modernen Mikroprozessoren war. Einer der Fortschritte der Delta-Regel gegenüber dem ursprünglichen Perceptron-Lernalgorithmus war ihre *Adaptivität*: War man weit von der richtigen Lösung entfernt, so veränderten sich auch die Verbindungsgewichte in größeren Schritten, die in der Nähe des Ziels kleiner werden – Nachteil: bei falscher Anwendung erhält man unendlich kleine Schrittweiten zum Ziel in der Nähe desselben. Während der späteren Flaute und aus Angst vor der wissenschaftlichen Unbeliebtheit der Neuronalen Netze wurde das ADALINE zwischenzeitlich in *Adaptive Linear Element* umbenannt – was später wieder rückgängig gemacht wurde.
- 1961** stellt KARL STEINBUCH Techniken assoziativer Speicherung vor, die als Vorgänger heutiger neuronaler Assoziativspeicher gesehen werden [Ste61]. Er beschreibt weiterhin Konzepte für neuronale Techniken und analysiert ihre Möglichkeiten und Grenzen.
- 1965** gibt NILS NILSSON in seinem Buch *Learning Machines* einen Überblick über die Fortschritte und Arbeiten dieser Periode der Erforschung Neuronaler Netze. Allgemein nimmt man an, die grundlegenden Prinzipien selbstlernender und damit landläufig gesprochen „intelligenter“ Systeme bereits entdeckt zu haben, was aus heutiger Sicht eine maßlose Überschätzung darstellt, aber damals für hohe Popularität und genügend Forschungsmittel sorgte.
- 1969** veröffentlichen MARVIN MINSKY und SEYMOUR PAPERT eine genaue mathematische Analyse des Perceptrons [MP69] um zu zeigen, dass das Perceptronmodell viele wichtige Probleme gar nicht repräsentieren kann (Stichwörter: *XOR-Problem* und *lineare Separierbarkeit*), und setzen so der Überschätzung, der Popularität und den Forschungsmitteln ein jähes Ende. Die weitergehende Folgerung, dass auch mächtigere Modelle die exakt gleichen Probleme aufweisen, verbunden mit der Prognose, dass das ganze Gebiet ein *research dead-end* sei, bewirken einen fast kompletten Rückgang der Forschungsgelder für die nächsten 15 Jahre, so unzutreffend diese Prognosen aus heutiger Sicht auch waren.





# Geschichte und Theorie – die Stille und Neuanfang

**1972** stellt TEUVO KOHONEN ein Modell des *linearen Assoziators*, eines Assoziativspeichermodells vor [Koh72], es wird im gleichen Jahr davon unabhängig von JAMES A. ANDERSON aus neurophysiologischer Sicht präsentiert [And72].


**1973** verwendet CHRISTOPH VON DER MALSBURG ein Neuronenmodell, was nichtlinear und biologisch besser motiviert ist [vdM73].

**1974** entwickelt PAUL WERBOS für seine Dissertation in Harvard das *Backpropagation of Error*-Lernverfahren [Wer74], das aber erst ein Jahrzehnt später seine heutige Bedeutung erlangt.

**1976 – 1980 und danach** werden von STEPHEN GROSSBERG viele Arbeiten (z.B. [Gro76]) vorgestellt, in denen eine Vielzahl von neuronalen Modellen mathematisch genau analysiert wird. Er widmet sich weiterhin ausführlich dem Problem, ein Neuronales Netz lernfähig zu halten, ohne bereits erlernte Assoziationen wieder zu zerstören – hieraus entstanden unter Mitarbeit von GAIL CARPENTER die Modelle der *Adaptive Resonance Theory*, kurz ART.

**1982** beschreibt TEUVO KOHONEN die nach ihm benannten *selbstorganisierenden Karten* (self organizing feature maps, SOM) [Koh82, Koh98] auf der Suche nach den Mechanismen der Selbstorganisation des Gehirns (er wusste, dass die Informationen über den Aufbau von Wesen im Genom gespeichert sind, das aber ganz wesentlich zu wenig Speicherplatz für eine Struktur wie das Gehirn besitzt – folglich muss sich das Gehirn zum Großteil selbst organisieren und aufbauen).

Weiterhin beschreibt JOHN HOPFIELD die nach ihm benannten Hopfieldnetze [Hop82], welche durch die Gesetze des Magnetismus in der Physik inspiriert sind. Sie erfuhren wenig technische Anwendungen, aber das Gebiet der Neuronalen Netze kam langsam wieder ins Rollen.

 **1983** wird von FUKUSHIMA, MIYAKE und ITO das neuronale Modell *Neocognitron* zur Erkennung handgeschriebener Zeichen vorgestellt [FMI83], welches eine Erweiterung des schon 1975 entwickelten Cognitrons darstellt.

**1985** veröffentlicht JOHN HOPFIELD einen Artikel, der Wege beschreibt, akzeptable Lösungen für das Travelling Salesman Problem durch *Hopfieldnetze* zu finden.

**1986** wird das Lernverfahren *Backpropagation of Error* als Verallgemeinerung der Delta-Regel durch die *Parallel Distributed Processing*-Gruppe separat entwickelt und weit publiziert [RHW86a]. Nicht linear separierbare Probleme wurden durch mehrschichtige Perceptrons lösbar, Marvin Minskys Negativabschätzungen waren mit einem Schlag widerlegt. Weiterhin machte sich zeitgleich in der Artificial Intelligence eine gewisse Ermüdung breit, verursacht durch eine Reihe von Fehlschlägen und untertroffenen Hoffnungen.



# Geschichte und Theorie – Neustart und Hype

ab ca. 1990:

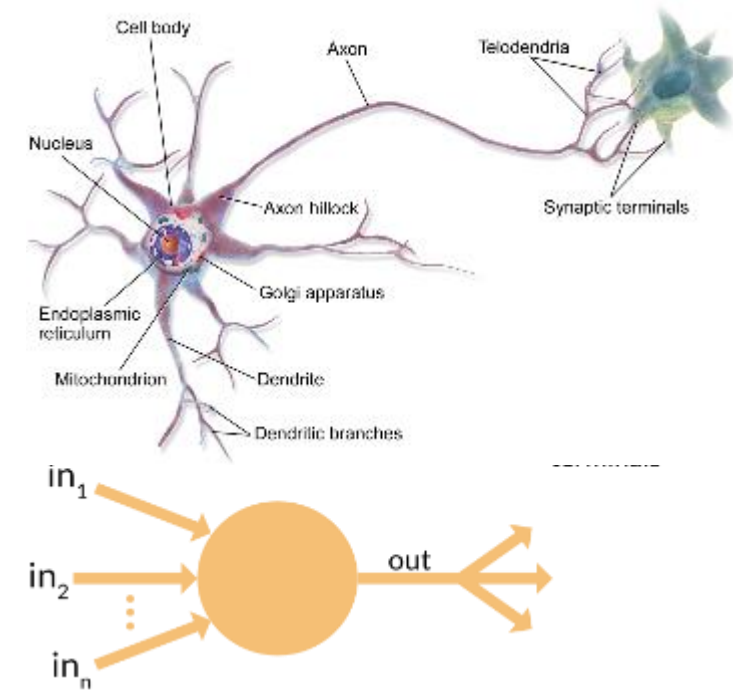
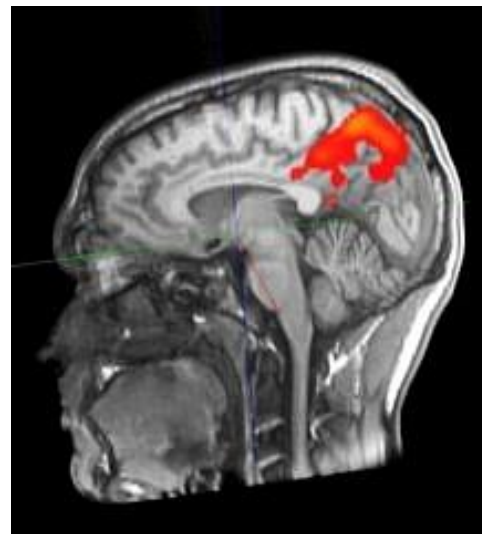
**George Hinton, Andrew Ng, ...**

Faktoren für die Renaissance von Neuronalen Netzen:

- a) grosse Rechenkraft günstig zur Verfügung (GPU)
- b) grosse Menge an Trainingsdaten
- c) (open source) libraries für die schnelle Entwicklung neuer Algorithmen

# Vorbild Gehirn

Vorbild Gehirn





# Vorbild Gehirn

Exkurs: MagnetResonanzImaging





# Vorbild Gehirn

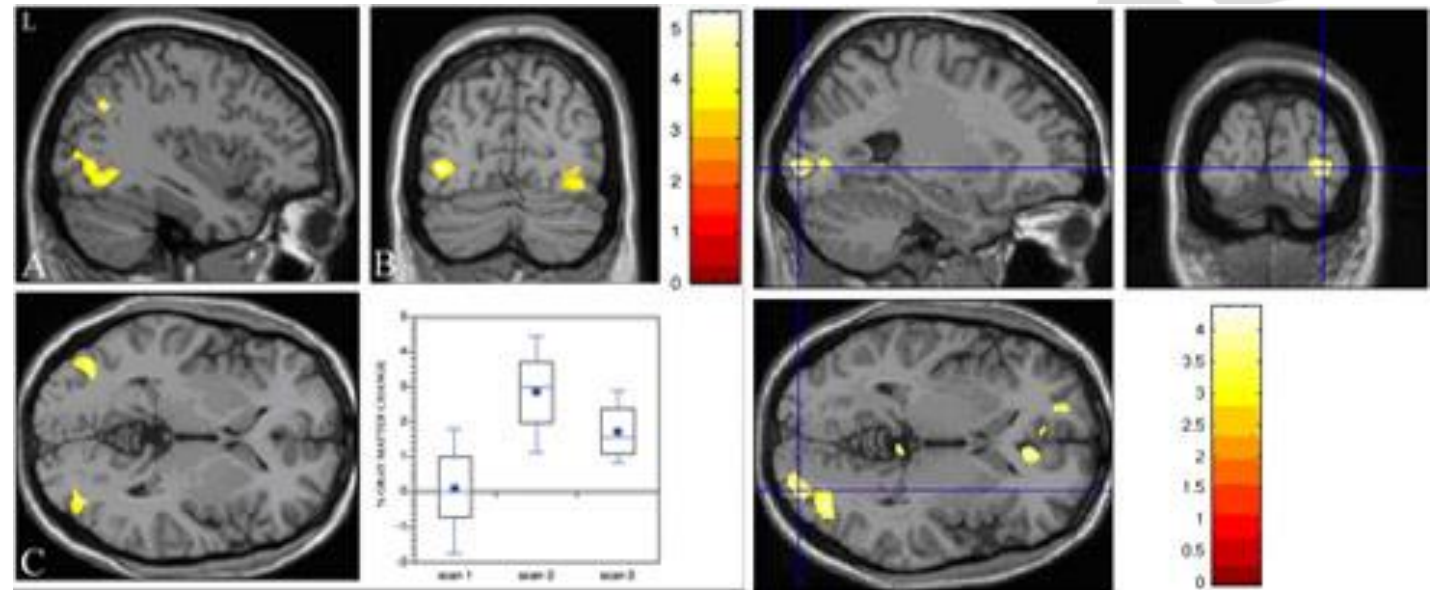
Die Graue Masse WÄCHST (2005)

## Jongleure im Hirnscanner

Um dieser Frage nachzugehen, bat eine Forschergruppe 44 Probanden (24 Frauen und 20 Männer) zwischen 50 und 67 Jahren, das Jonglieren zu erlernen. Die Versuchsteilnehmer wurden vor und nach dem dreimonatigen Training sowie nach einer dreimonatigen Trainingspause mit Hilfe der 3-Tesla-Kernspintomografie untersucht. Verglichen wurden diese Daten mit den Hirnen von 25 untrainierten Personen (17 Frauen und acht Männern) zwischen 55 und 67 Jahren, die an denselben Tagen gescannt wurden.

Nach der Trainingsphase ließ sich bei den Jongleuren eine einseitige Vergrößerung der grauen Substanz im visuellen Assoziationscortex erkennen. Diese Gehirnregion ist darauf spezialisiert, Bewegung im Raum wahrzunehmen. Nach der dreimonatigen Trainingspause hatte sich die Erweiterung teilweise wieder zurückgebildet. Die Kontrollgruppe zeigte keinerlei Veränderungen in diesem Bereich (Boyke, 2008).

Ausschließlich bei den Jongleuren fanden die Forscher zudem eine Vergrößerung im Hippocampus, der Hirnregion, die für das Lernen wichtig ist. Darüber hinaus zeigten sich Vergrößerungen im Nucleus accumbens, der zum hirneigenen Belohnungssystem gehört. Vom Hippocampus weiß man, dass sich dort neue Nervenzellen bilden können.





# Vorbild Gehirn

## Neuere Erkenntnisse

### Marihuana-Forschung

### Cannabinoid lässt graue Zellen wachsen

Kann Marihuana gegen Depressionen helfen? Bei Mäusen hatte eine synthetische Droge aus der Cannabisfamilie jedenfalls erheblichen Einfluss auf das Wachstum der grauen Zellen.



Samstag, 15.10.2005 10:38 Uhr

[Drucken](#) [Nutzungsrechte](#) [Feedback](#)

05. Oktober 2016 ■ Biowissenschaften Hirnforschung Kognitionswissenschaft Verhaltensforschung Zool

## Gähnen stimuliert das Hirn

Von Joachim Czichos

**Je größer und komplexer das Gehirn, desto länger gähnt ein Säugetier – wahrscheinlich um dadurch die Durchblutung des Denkkorgans zu verbessern**



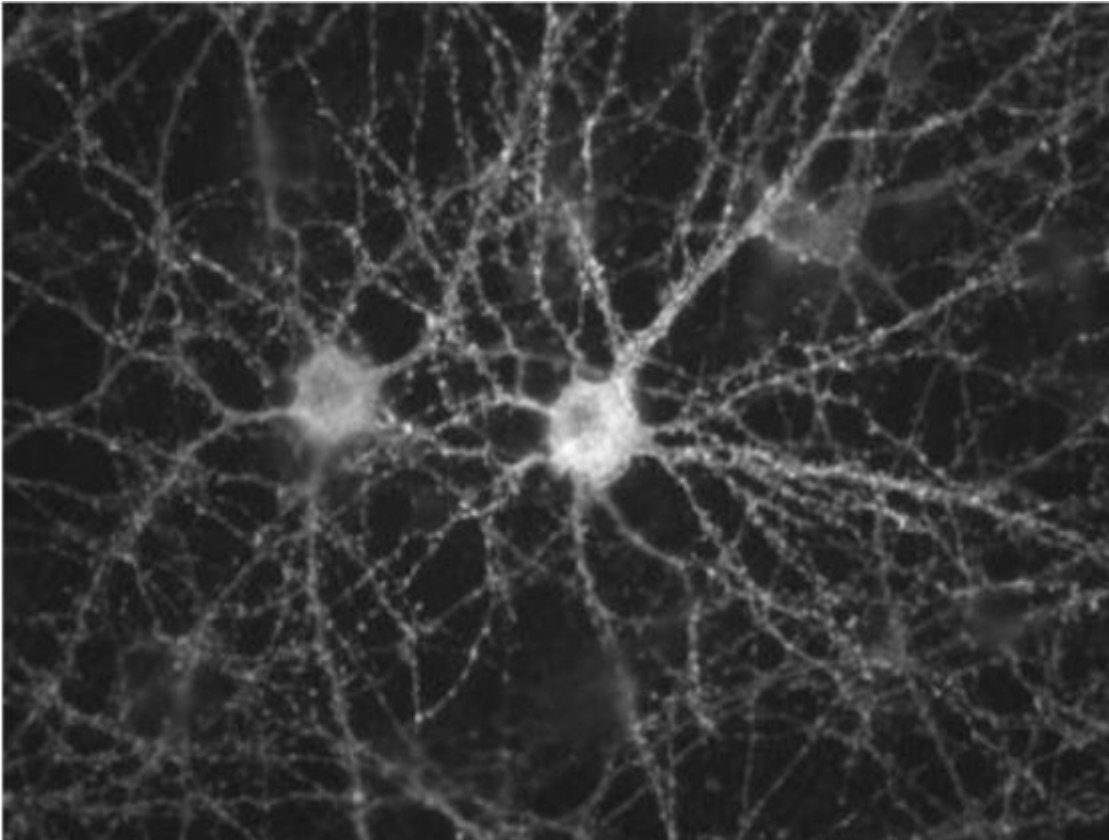
Die Gähndauer des Löwen liegt zwischen der von Kapuzineraffe und Pferd.

Oneonta (USA) - Gähnen ist ein Verhalten, das sich bei zahlreichen Wirbeltieren entwickelt und im Lauf der Evolution bis hin zum Menschen erhalten hat. Es muss also eine wichtige Funktion erfüllen, die aber noch immer nicht eindeutig geklärt ist. Jetzt haben amerikanische Psychologen einen aufschlussreichen Zusammenhang entdeckt: Demnach sind bei verschiedenen Säugetieren die Neuronenzahl in der Großhirnrinde und das Hirngewicht umso größer, je länger das Gähnen bei einer Tierart durchschnittlich dauert. Das unterstützt die Vermutung, dass das Gähnen kognitive Hirnfunktionen fördern könnte, indem es die Durchblutung und Kühlung des Denkkorgans verstärkt, schreiben die Forscher im Fachblatt „Biology Letters“. Ob besonders intelligente Menschen auch besonders lange gähnen, wurde noch nicht untersucht.



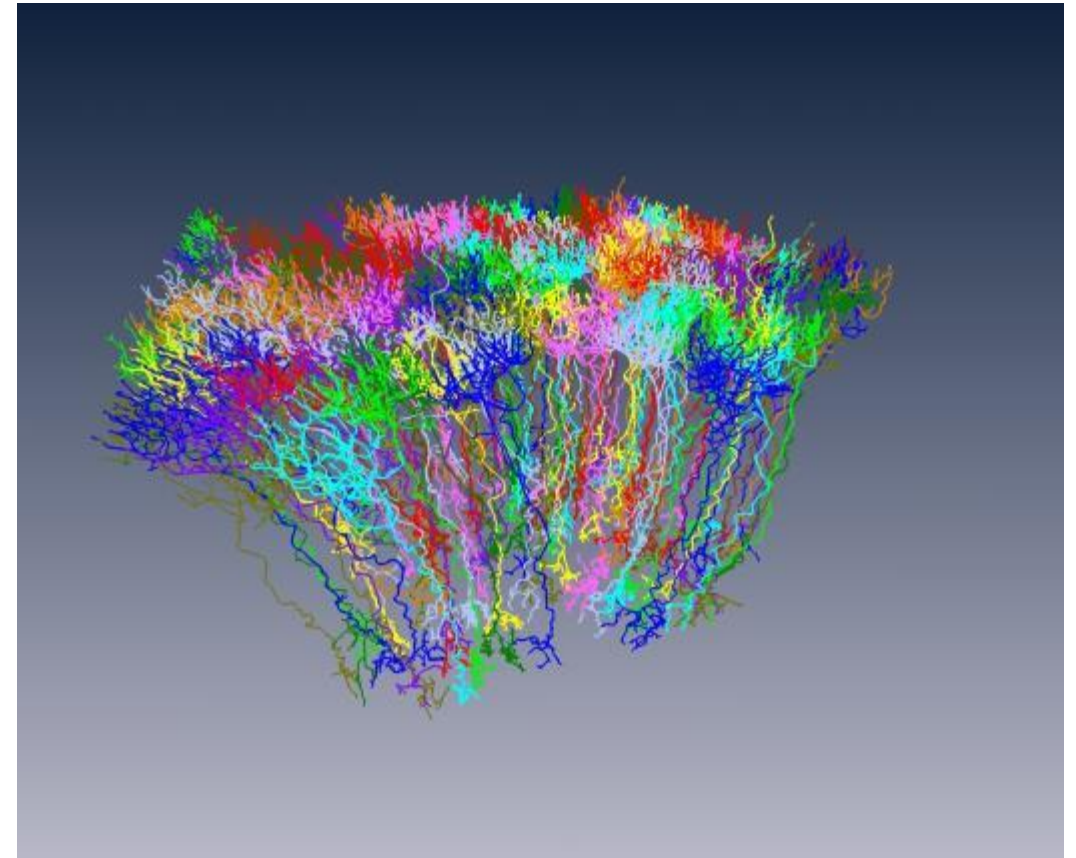


# Gehirn vs. Computer



Nervenzellen nutzen meist chemische Synapsen  
An einem Neuron können hunderte bis tausende  
Synapsen angedockt sein.

Rekonstruktion von 114 bipolaren Nervenzellen  
von einem Stück Mausretina





# Geschichte und Theorie

Neuere Erkenntnisse

	PC 2016	Menschliches Gehirn
Recheneinheiten	12 Kerne, $10^9$ Transistoren	$10^{11}$ Neuronen
Speichereinheiten	$10^{11}$ Bits RAM	$10^{11}$ Neuronen
	$10^{13}$ Bits Festplatte	$10^{14}$ Synapsen
Zyklus	$10^{-9}$ Sekunden	$10^{-3}$ Sekunden
Operationen / Sekunde	$10^{10}$	$10^{17}$
Speicherupdates / Sekunde	$10^{10}$	$10^{14}$





# Gehirn vs. Computer

- Innerhalb eines Neurons wird ein einkommendes Signal elektrisch weitergeleitet.
- Zwischen zwei Neuronen werden Signale in der Regel chemisch über Neurotransmitter übertragen.
- Die elektrische Weiterleitung funktioniert nach dem Alles-oder-Nichts-Prinzip: Erst wenn die Stärke des Signals einen Schwellenwert übersteigt, wird im Axon das Aktionspotenzial generiert.
- Dabei helfen die Synapsen, die das elektrische Signal des Aktionspotenzials in ein chemisches “übersetzen”: Sie setzen Botenstoffe, Neurotransmitter, in den Spalt zwischen Sender- und Empfängerzelle frei.
- Die Empfängerzelle kann die Neurotransmitter über Rezeptoren aufnehmen und in ein elektrisches Signal, das postsynaptische Signal, übersetzen.
- Die Botschaft und Dringlichkeit eines Signals zeigt sich an der Anzahl und der Frequenz der Aktionspotenziale.



# Geschichte und Theorie

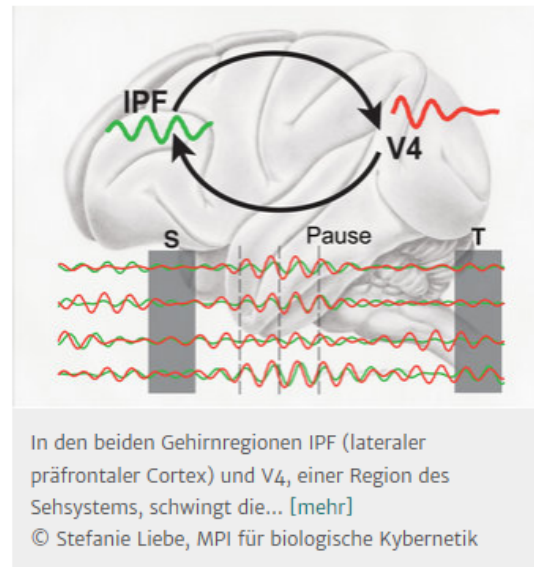
## Das Kurzzeitgedächtnis schwingt synchron

Wissenschaftler haben entschlüsselt, wie das Kurzzeitgedächtnis funktioniert

19. SEPTEMBER 2014

Gehirn Neurobiologie

Schüler und Studenten sind oft große Fans des Kurzzeitgedächtnisses – wenn sie nämlich am Tag vor einer Prüfung große Mengen Unterrichtsstoff einpauken müssen. Obwohl es nur kurz vorhält, ist das Kurzzeitgedächtnis ein komplexes Netzwerk aus Nervenzellen im Gehirn, das verschiedene Gehirnregionen umfasst. Beim Speichern müssen diese Gebiete zusammenarbeiten. Forscher vom Max-Planck-Institut für biologische Kybernetik in Tübingen haben nun herausgefunden, dass die beteiligten Regionen synchron miteinander aktiv sein müssen, damit wir uns kurzzeitig an etwas Gesehenes erinnern.



Wenn wir etwas sehen, werden die Signale aus den Augen in Gebieten der Großhirnrinde am Hinterkopf verarbeitet. Für das Kurzzeitgedächtnis müssen dagegen Regionen im vorderen Teil des Großhirns aktiv sein. Damit wir uns an etwas, das wir gesehen haben, für eine kurze Zeit erinnern können, müssen also diese weit voneinander entfernten Hirnteile ihre Informationen zusammenführen.

Wie das funktioniert, lässt sich nur an Affen untersuchen. Wissenschaftler der Abteilung von Nikos Logothetis am Max-Planck-Institut für biologische Kybernetik in Tübingen haben die elektrische Aktivität in einer Sehregion und im vorderen Gehirnbereich gemessen, während sich die Tiere an verschiedene Bilder erinnern mussten.

Die Wissenschaftler haben dabei in beiden Gehirnregionen elektrische Schwingungen beobachtet, sogenannte Theta-Band-Schwingungen. Überraschenderweise treten diese Schwingungen nicht unabhängig voneinander auf, sondern synchron. Je stärker die Regionen synchron

aktiv sind, desto besser konnten sich die Tiere an ein Bild erinnern.



# Neuronale Netze - Begriffe

- Perzeptron (<https://appliedgo.net/perceptron/>)
- Hebb'sche Regel
- ReLu und Varianten, Sigmoid, Logit
- Backpropagation ([playground.tensorflow.org](https://playground.tensorflow.org))
- Optimizer
- Vanishing/Exploding Gradient
- Convolutional Layer
- Max Pooling
- Softmax
- Cross Entropy
- Dropout
- Transfer Learning



# Neuronale Netze - Begriffe

**Initialization**

He initialization

**Activation function**

ELU

**Normalization**

Batch Normalization

**Regularization**

Dropout

**Optimizer**

Nesterov Accelerated Gradient

**Learning rate schedule**

None

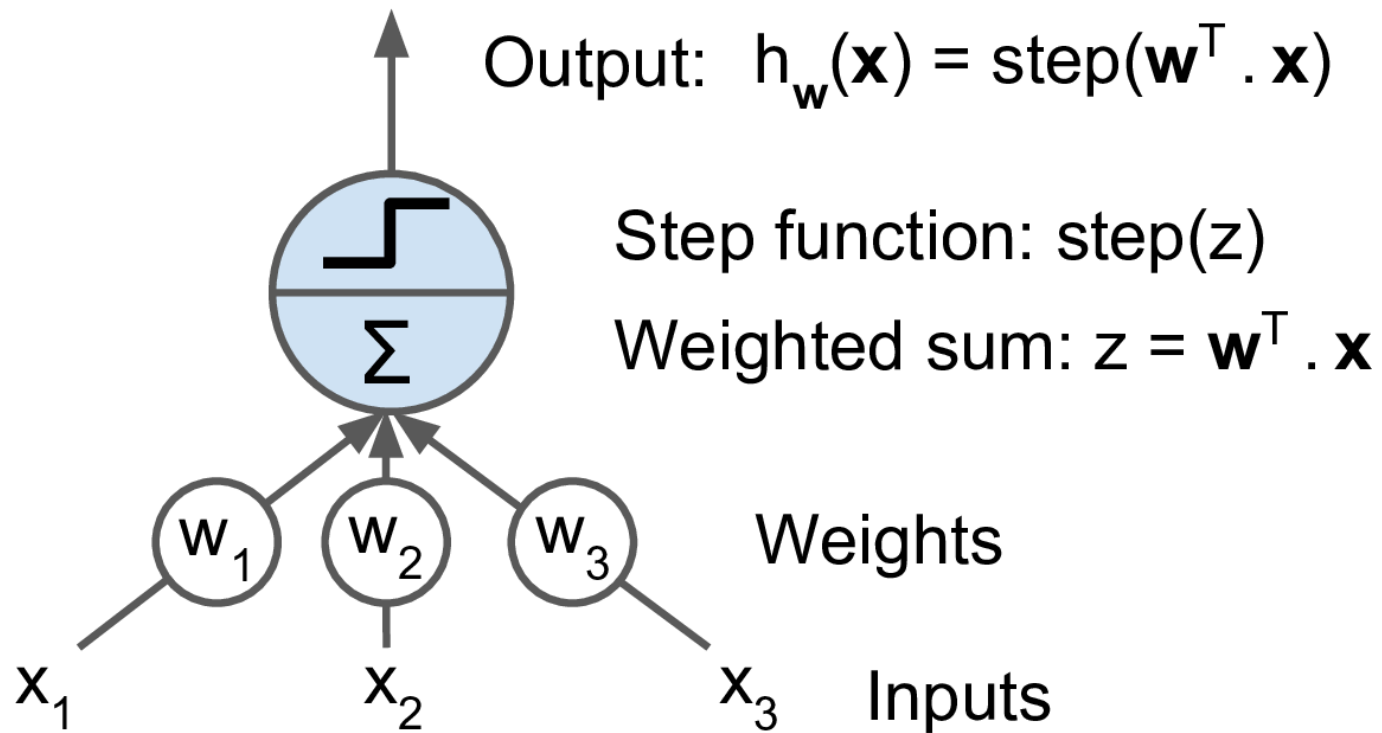
Aurelion Geron





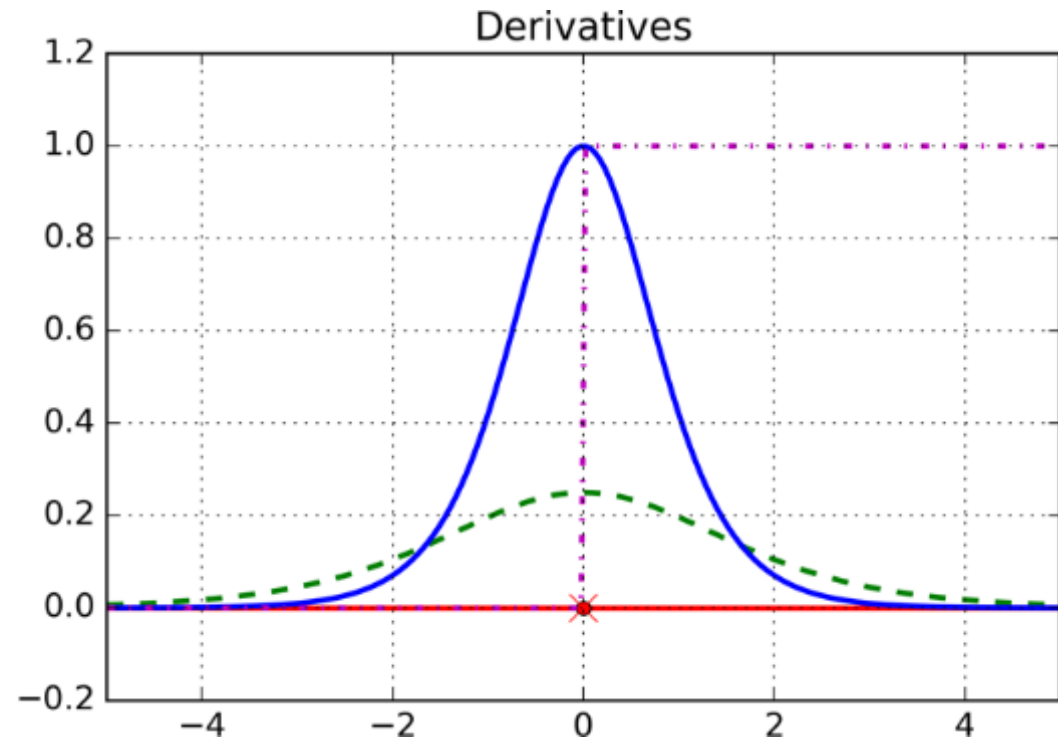
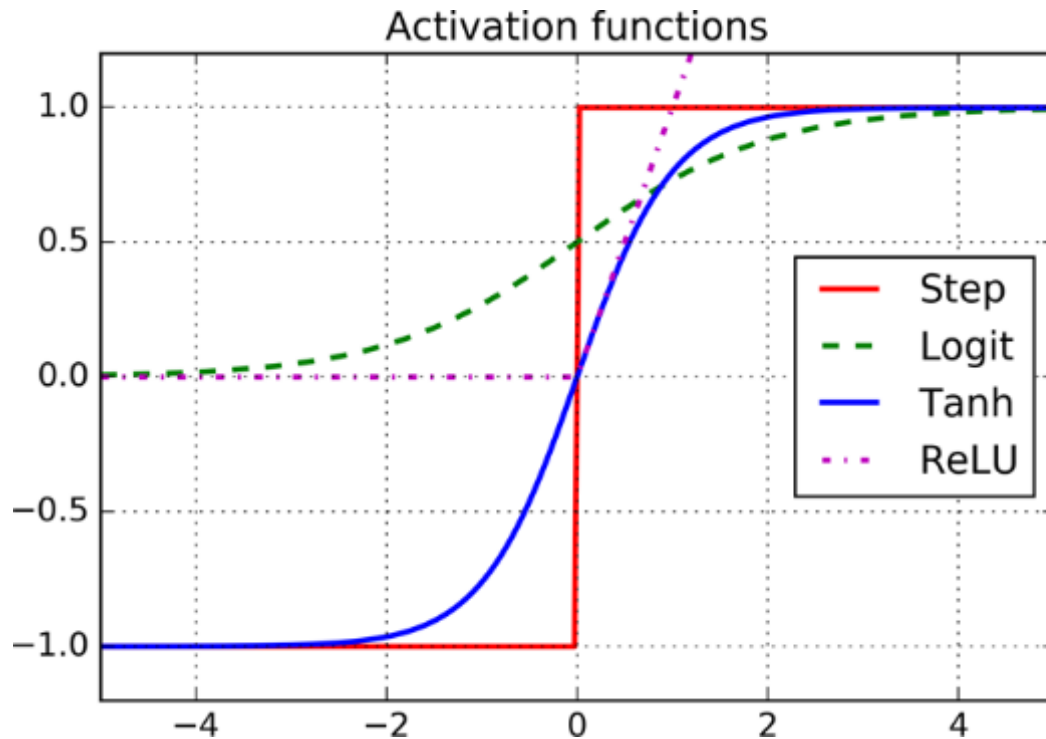
# Begriffe - Perzeptron

Perzeptron





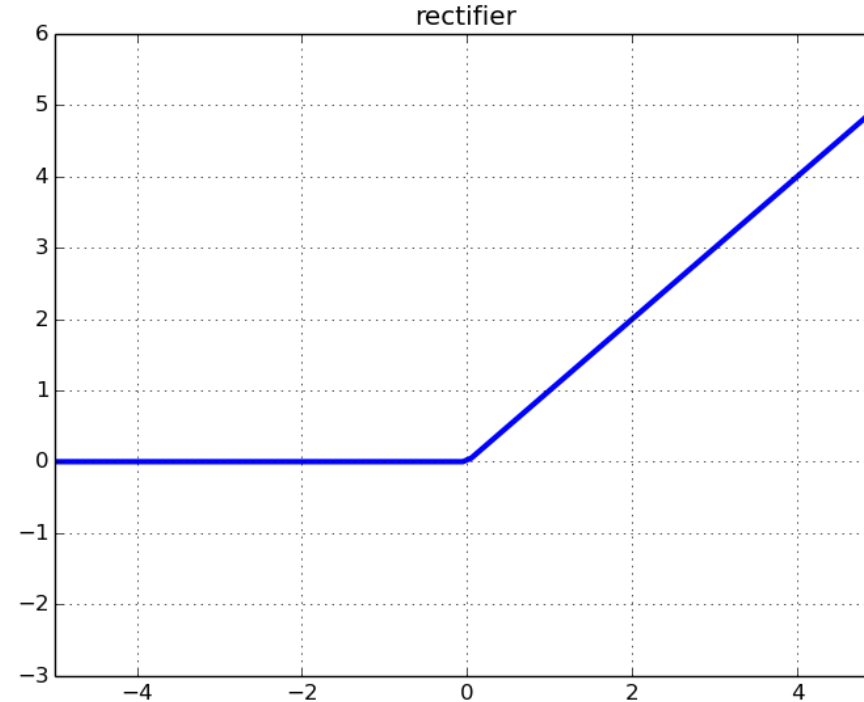
# Begriffe – Activation Function





# Begriffe – Activation Function

## ■ ReLu Function





# Begriffe – Hebb'sche Lernregel

## ■ Hebb'sche Lernregel

$$\Delta w_{ij} = \eta \cdot a_i \cdot o_j$$

mit

- $\Delta w_{ij}$ : Veränderung des Gewichtes von Neuron i zu Neuron j (also die Änderung der Verbindungsstärke dieser beiden Neuronen)
- $\eta$ : *Lernrate* (ein geeignet zu wählender konstanter Faktor)
- $a_i$ : Aktivierung von Neuron i
- $o_j$ : Ausgabe von Neuron j, das mit Neuron i verbunden ist.





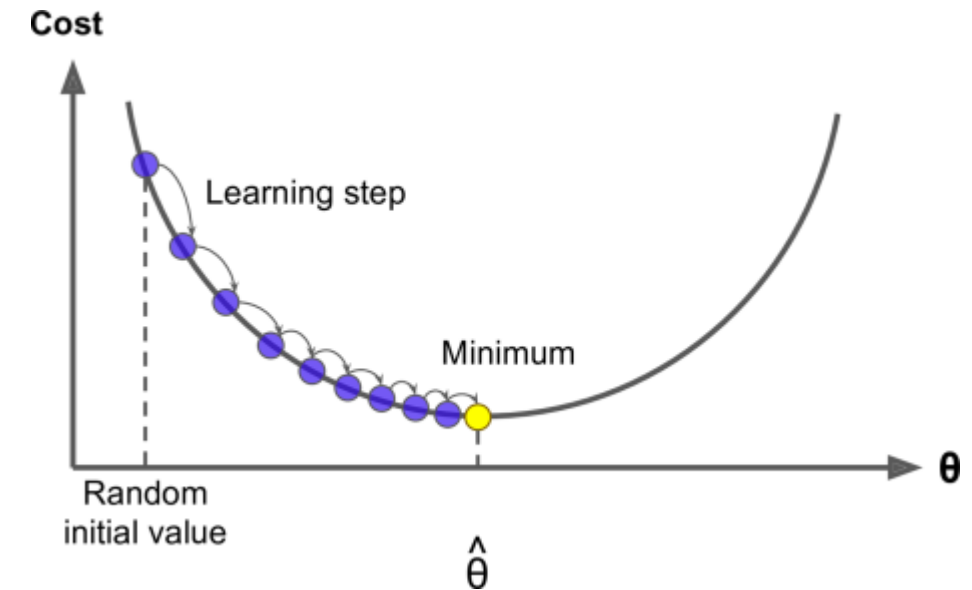
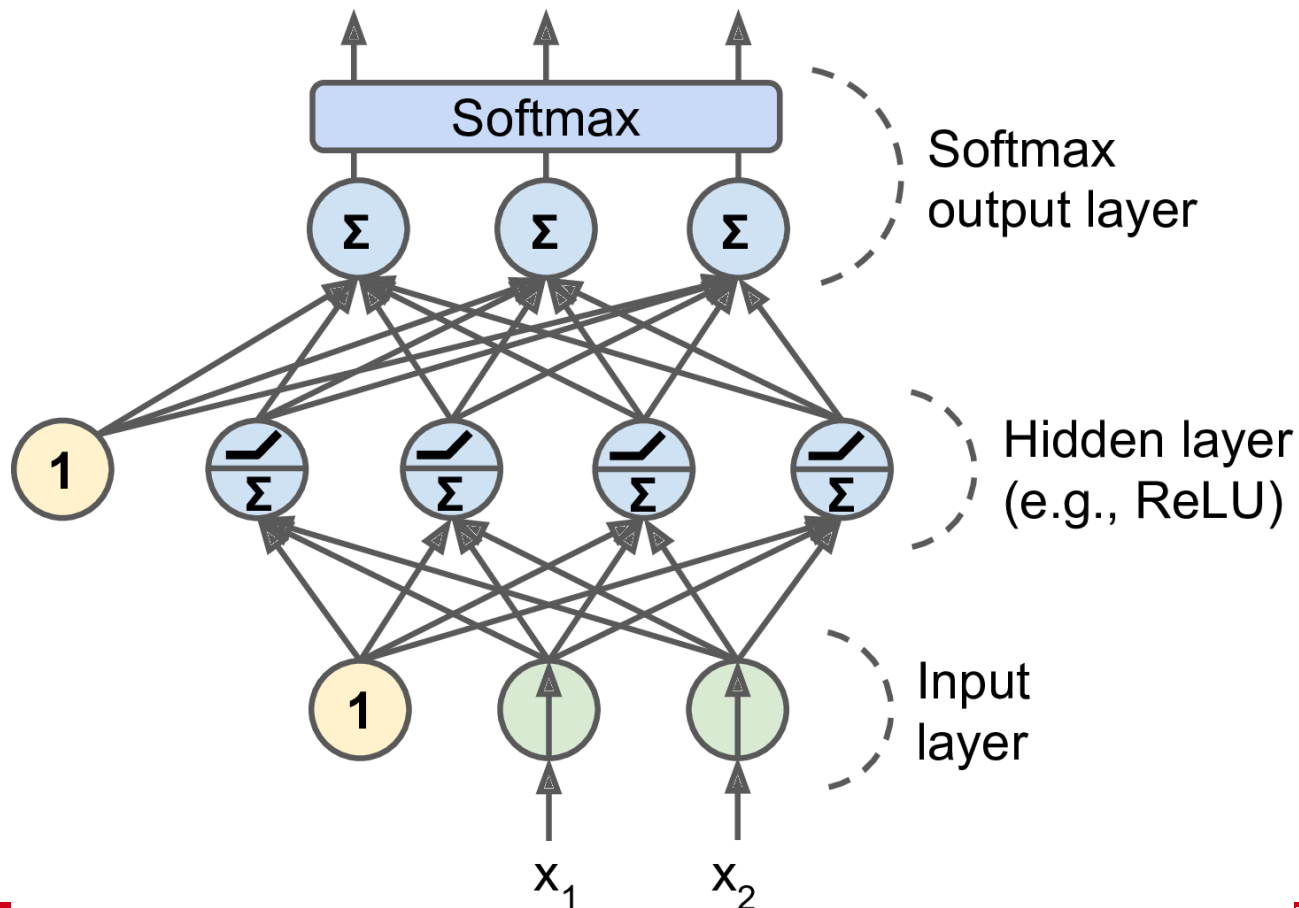


# Begriffe – Backpropagation

## ■ Backpropagation

(<https://youtu.be/llg3gGewQ5U> - 3Blue1Brown Video)

(<https://youtu.be/tleHLnjs5U8> - 3Blue1Brown Video)



# Begriffe – Optimization

## ■ Optimizer

1. Momentum Optimization

2. Nesterov Accelerated Gradient

3. RMSProp

4. AdaGrad

5. Adam Optimization (**Ad**aptiv **M**ovement)

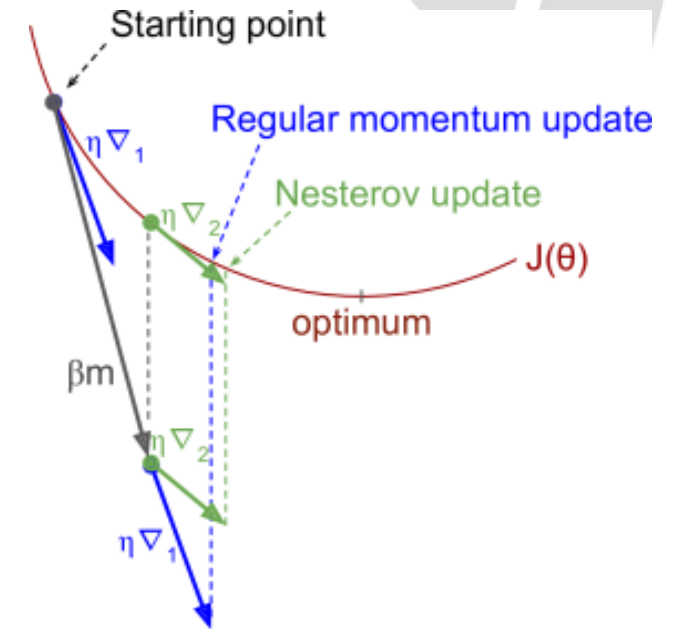
Kombination von Momentum und RMSProp

$$1. \quad \mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\theta} J(\theta)$$

$$2. \quad \theta \leftarrow \theta + \mathbf{m}$$

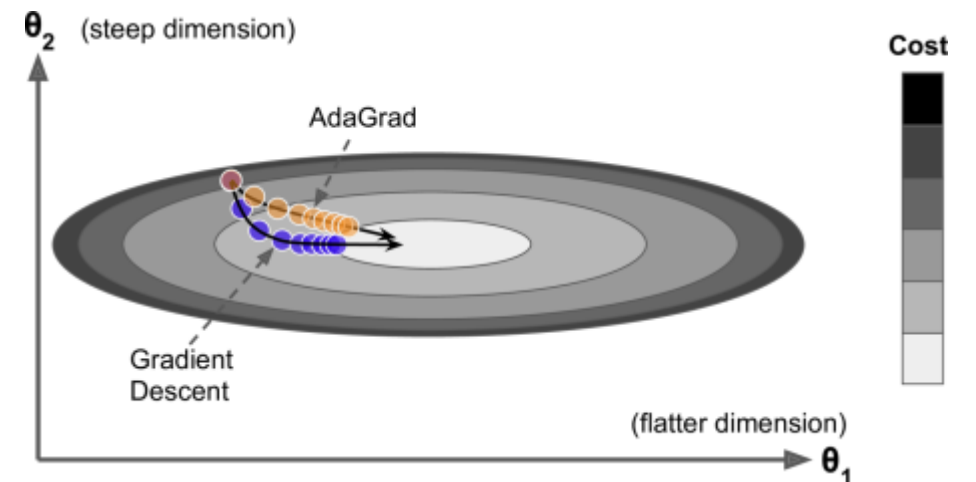
$$1. \quad \mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\theta} J(\theta + \beta \mathbf{m})$$

$$2. \quad \theta \leftarrow \theta + \mathbf{m}$$



## WARNING

This book initially recommended using Adam optimization, because it was generally considered faster and better than other methods. However, a 2017 [paper](#)<sup>16</sup> by Ashia C. Wilson et al. showed that adaptive optimization methods (i.e., AdaGrad, RMSProp and Adam optimization) can lead to solutions that generalize poorly on some datasets. So you may want to stick to Momentum optimization or Nesterov Accelerated Gradient for now, until researchers have a better understanding of this issue.





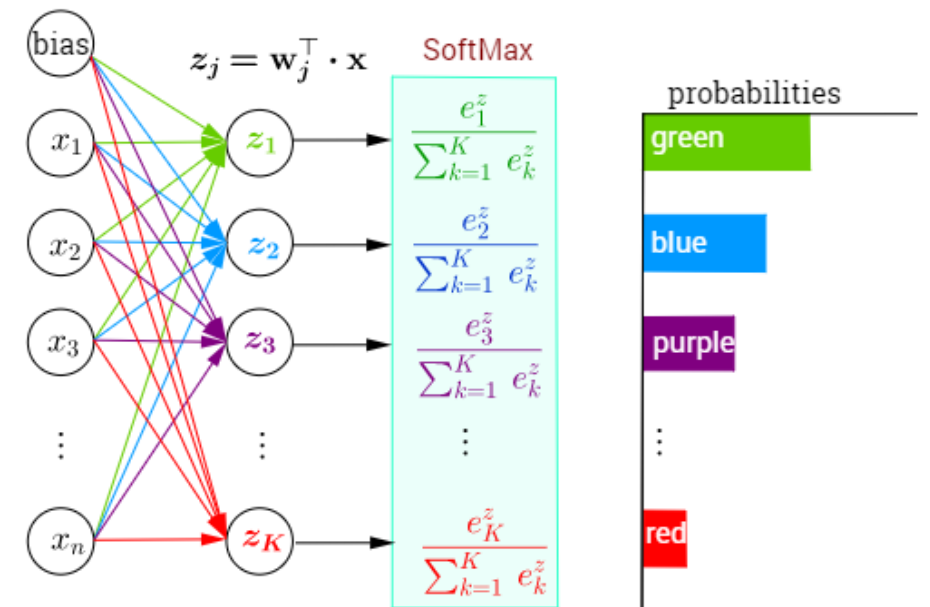
# Begriffe - Softmax

## ■ Softmax

Warum Softmax?

1. Die Ableitung der Softmax-Funktion ist softmax(1-softmax)  
Hilfreich bei Gradientenbasierten Lernen
2. Output im Bereich [0-1]
3. Output vector summiert sich auf 1

$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$





# Begriffe – Cross Entropy

## ■ Cross Entropy

$$C = -\frac{1}{n} \sum_x [y \ln a + (1 - y) \ln(1 - a)],$$

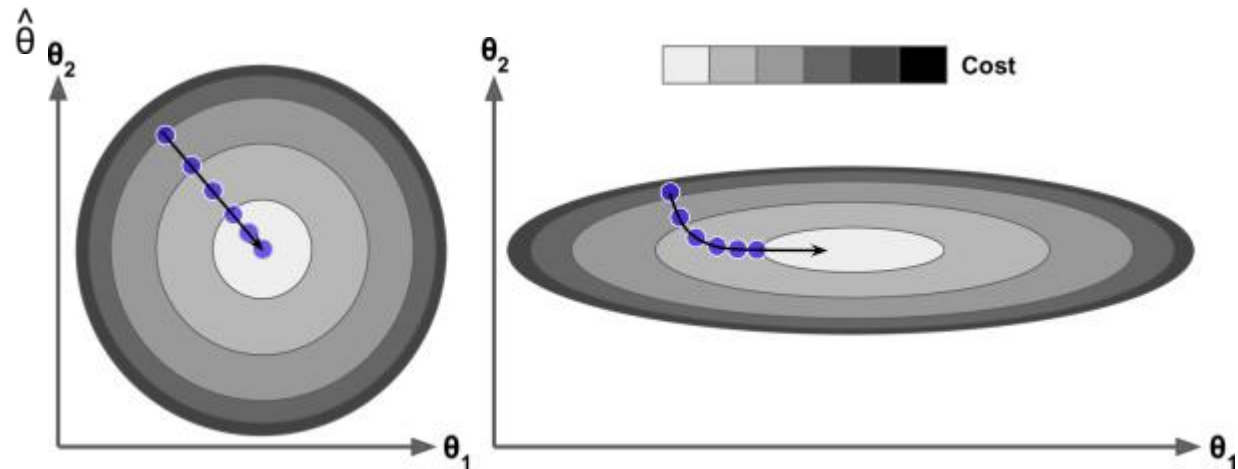
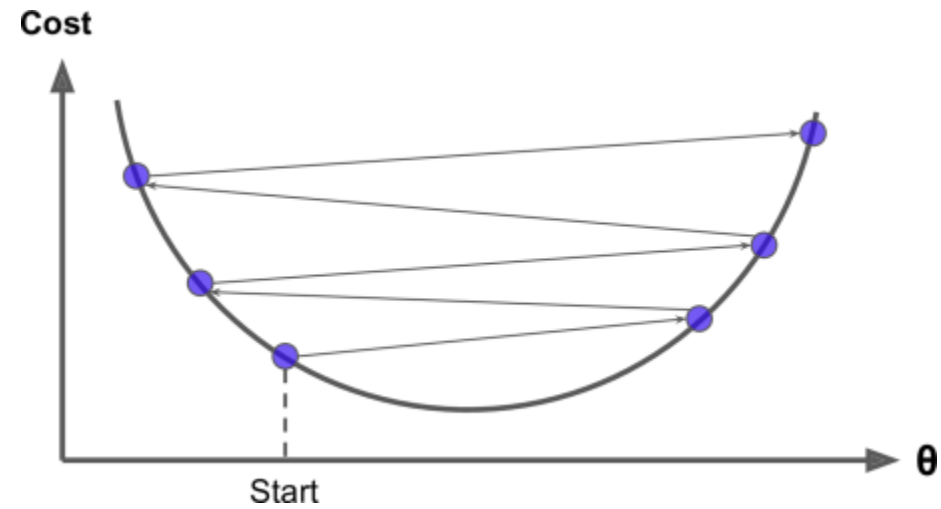
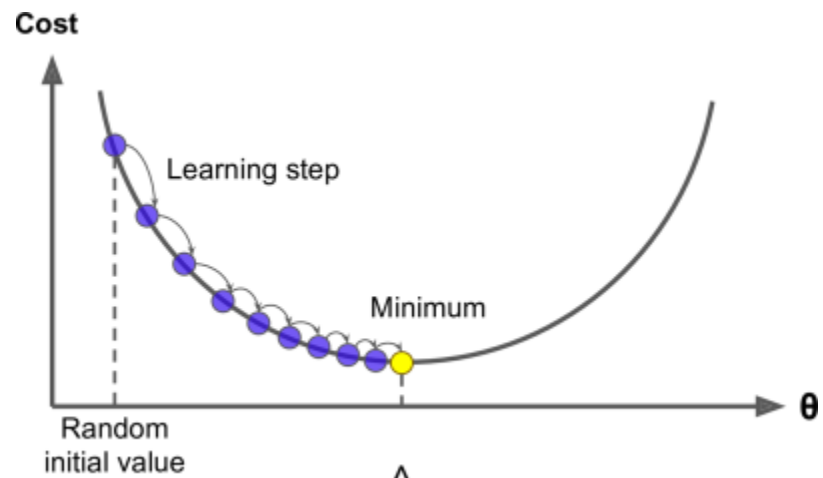
$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x \frac{\sigma'(z) x_j}{\sigma(z)(1 - \sigma(z))} (\sigma(z) - y).$$

$$\frac{\partial C}{\partial w_j} = \frac{1}{n} \sum_x x_j (\sigma(z) - y).$$





## ■ Vanishing/Exploding Gradient





# Begriffe - Vanishing/Exploding Gradient

## ■ Vanishing/Exploding Gradient

Lösungen:

1. Multi-level hierarchy  
pre-training einer Ebene/Layers durch unsupervised learning (Jürgen Schmidhuber 1992)
2. "Richtige" Initialisierung der Gewichte
3. Verwendung von ReLu



# Begriffe - Initialisierung

## Perzeptron

Activation function	Uniform distribution $[-r, r]$	Normal distribution
Logistic	$r = \sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
Hyperbolic tangent	$r = 4\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = 4\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$
ReLU (and its variants)	$r = \sqrt{2}\sqrt{\frac{6}{n_{\text{inputs}} + n_{\text{outputs}}}}$	$\sigma = \sqrt{2}\sqrt{\frac{2}{n_{\text{inputs}} + n_{\text{outputs}}}}$

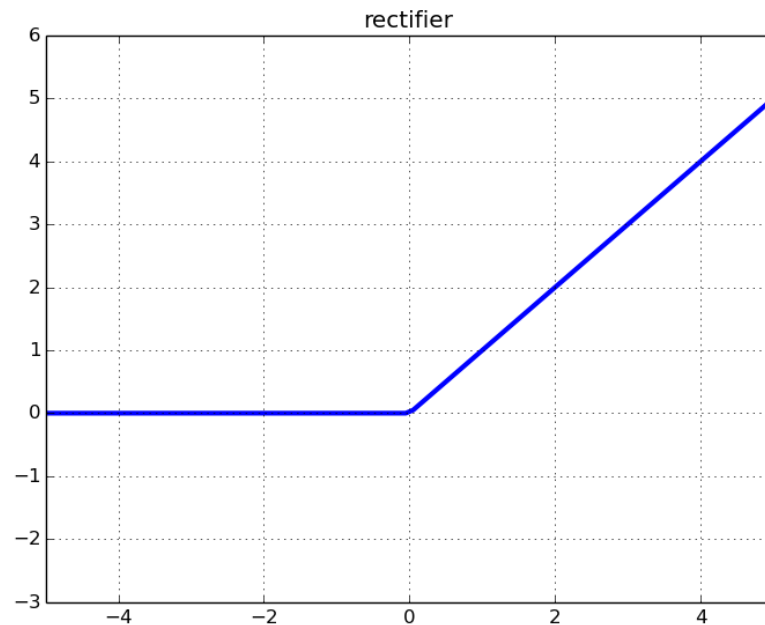
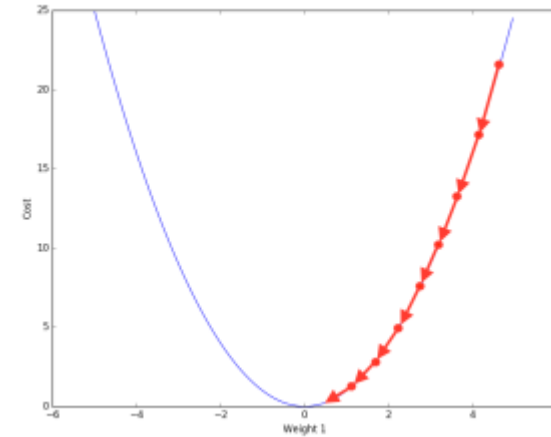
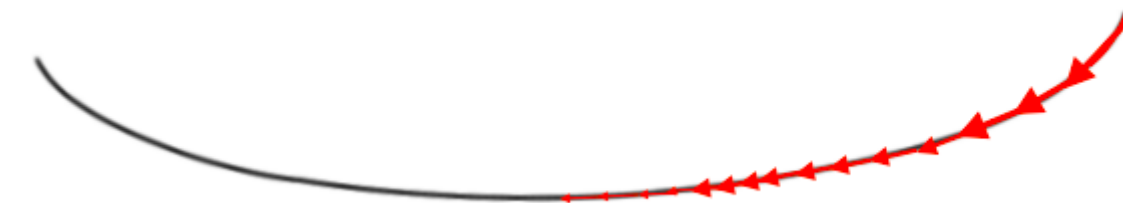
By default, the `tf.layers.dense()` function (introduced in [Chapter 10](#)) uses Xavier initialization (with a uniform distribution). You can change this to He initialization by using the `variance_scaling_initializer()` function like this:

```
he_init = tf.contrib.layers.variance_scaling_initializer()
hidden1 = tf.layers.dense(X, n_hidden1, activation=tf.nn.relu,
                           kernel_initializer=he_init, name="hidden1")
```

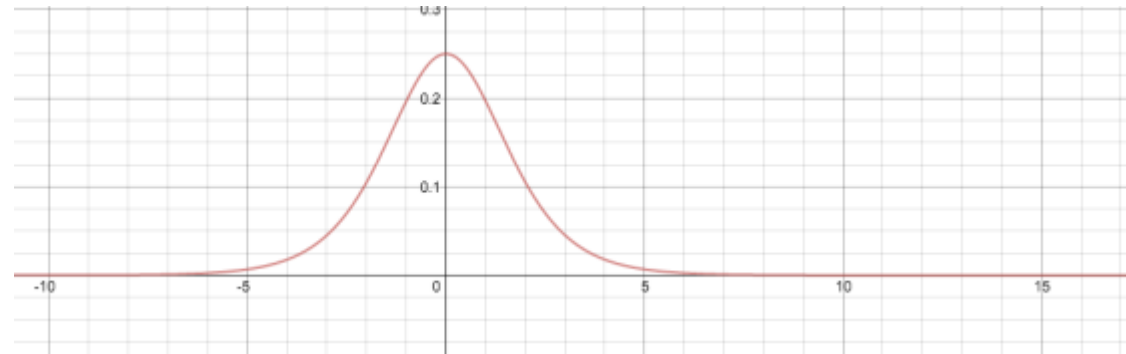


# Begriffe – Vanishing/Exploding Gradient

## ■ Vanishing/Exploding Gradient



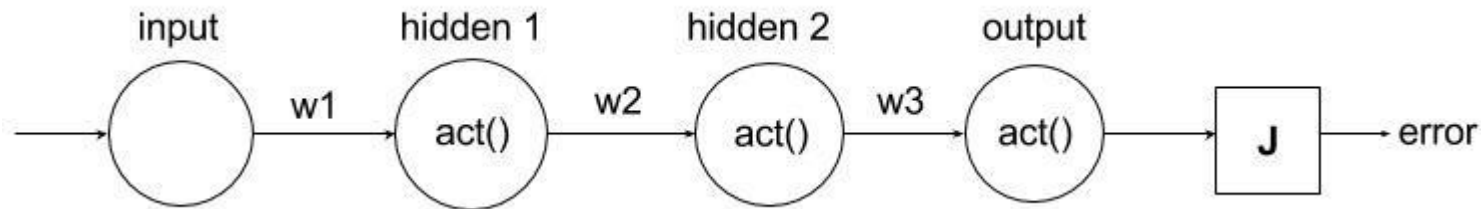
$$= S(1 - S)$$





# Begriffe – Vanishing/Exploding Gradient

## ■ Vanishing/Exploding Gradient



$$\frac{\partial error}{\partial w1} = \frac{\partial error}{\partial output} * \frac{\partial output}{\partial hidden2} * \frac{\partial hidden2}{\partial hidden1} * \frac{\partial hidden1}{\partial w1}$$

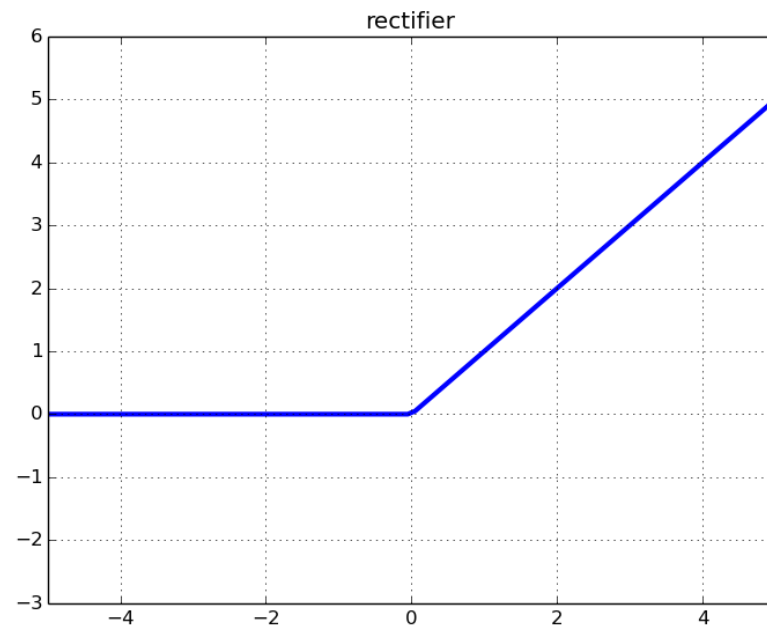


# Begriffe – Vanishing/Exploding Gradient

## ■ Vanishing/Exploding Gradient

$$RELU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

$$RELU(x) = \max(0, x)$$



$$\frac{d}{dx} 0 = 0$$

$$\frac{d}{dx} x = 1$$

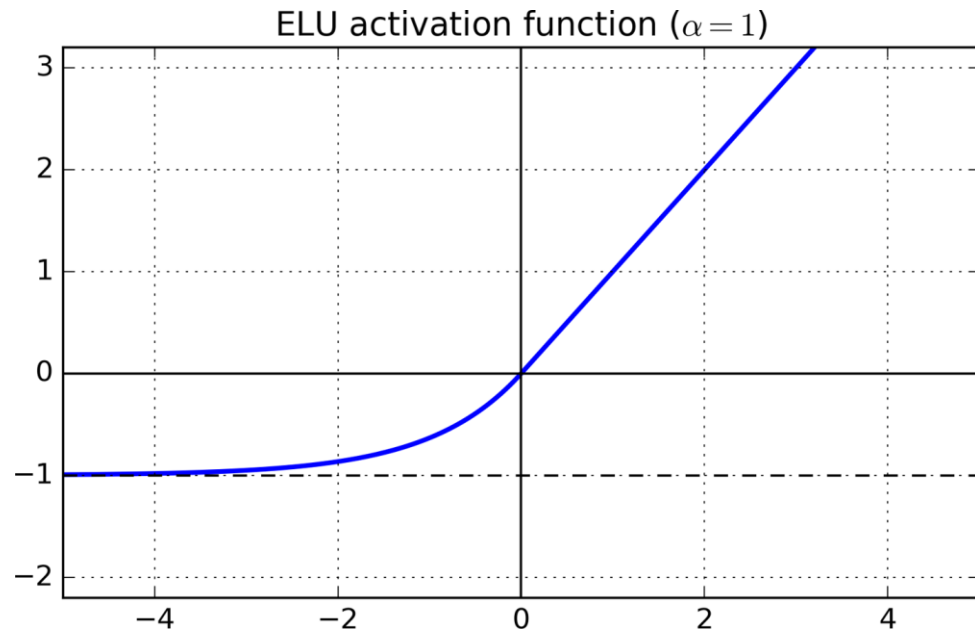
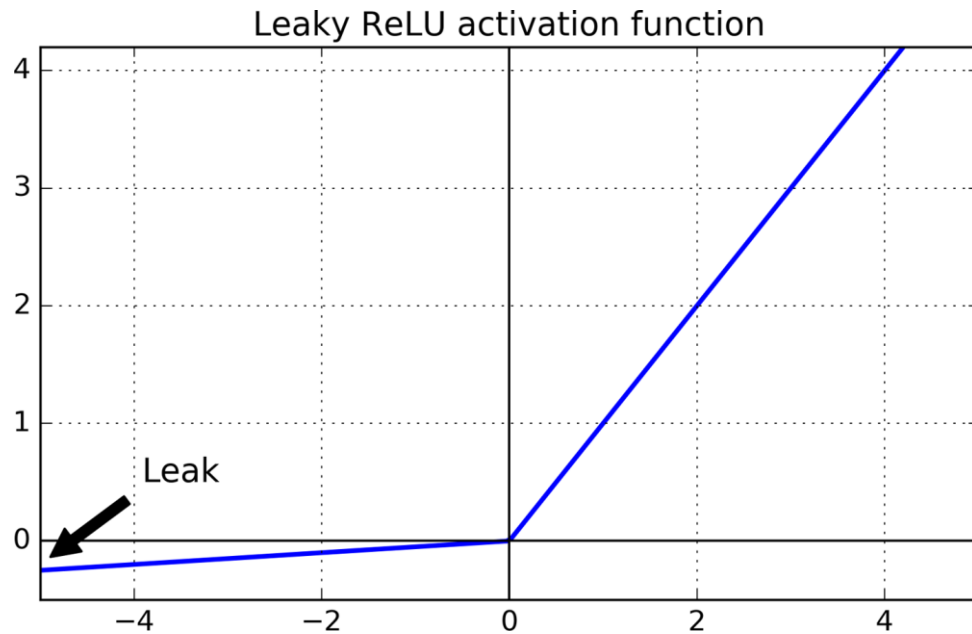
$$\frac{d}{dx} RELU(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$





# Begriffe – Vanishing/Exploding Gradient

## ■ Dying ReLu Problem





■ Vanishing/Exploding Gradient

$$\frac{\partial}{\partial \theta_j} \text{MSE}(\theta) = \frac{2}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^{(i)} - y^{(i)}) x_j^{(i)}$$

$$\nabla_{\theta} \text{MSE}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{MSE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{MSE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{MSE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\mathbf{X} \cdot \theta - \mathbf{y})$$



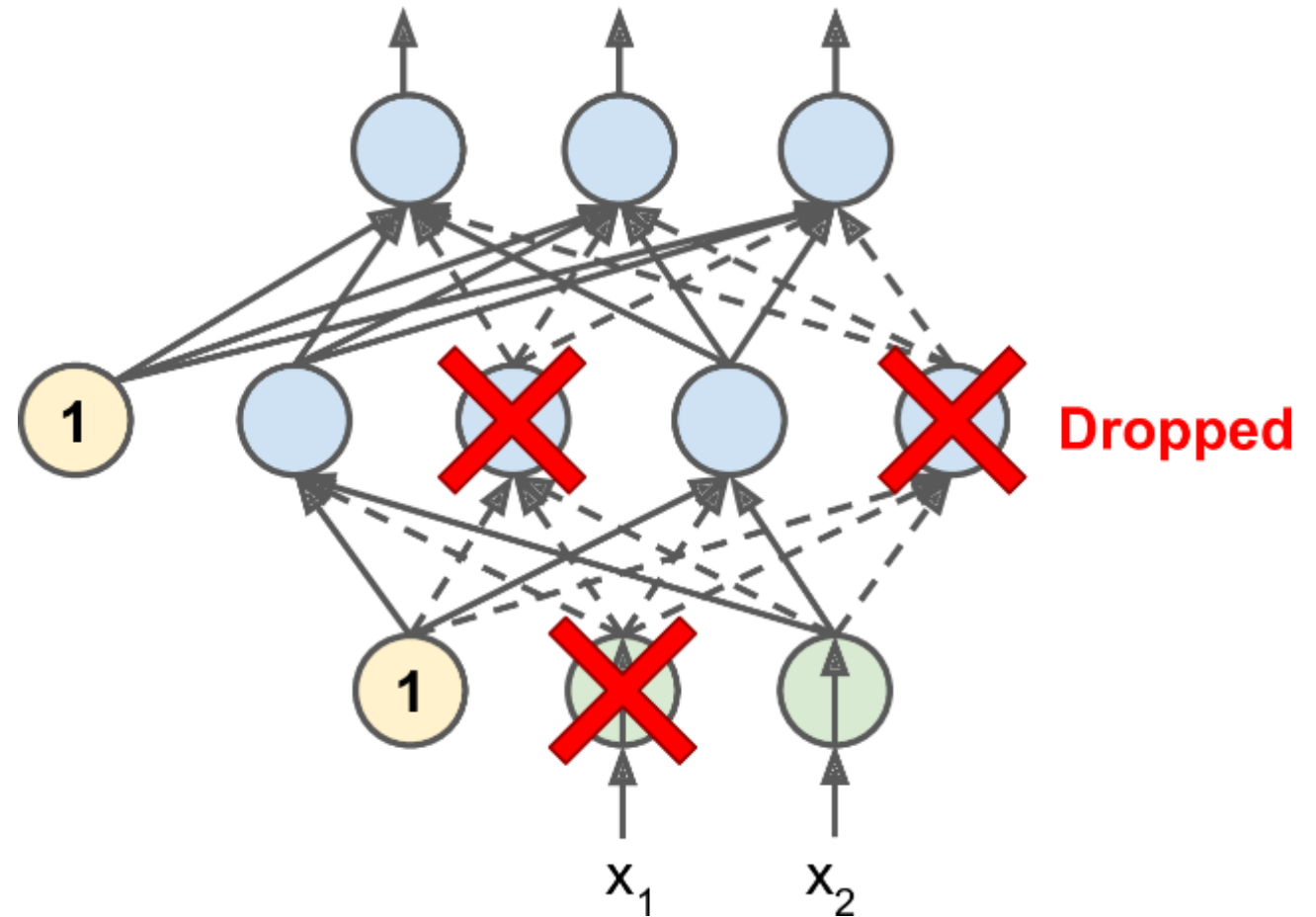
# Begriffe - DropOut

## ■ Dropout

Zur Vermeidung von Overfitting

oder bekannte Methoden

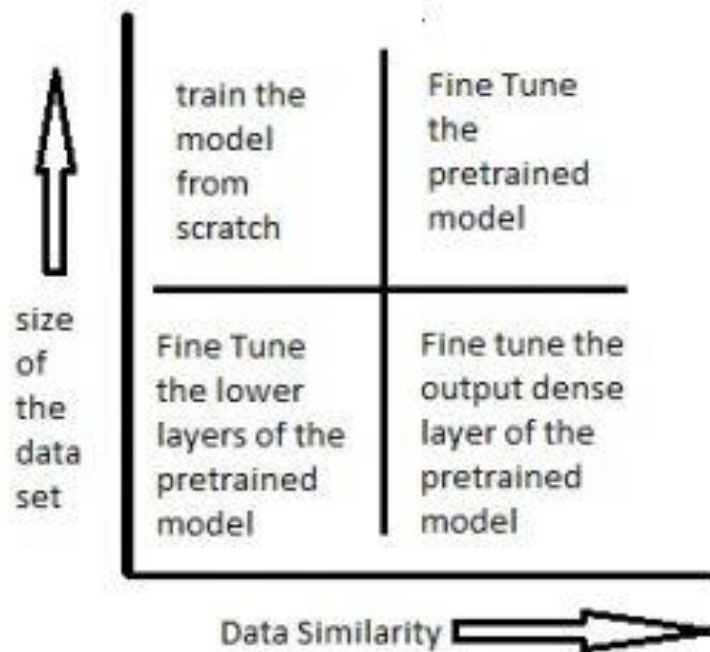
1. L1, L2 – Regularization
2. Early Stopping





# Begriffe – Transfer Learning

## ■ Transfer Learning

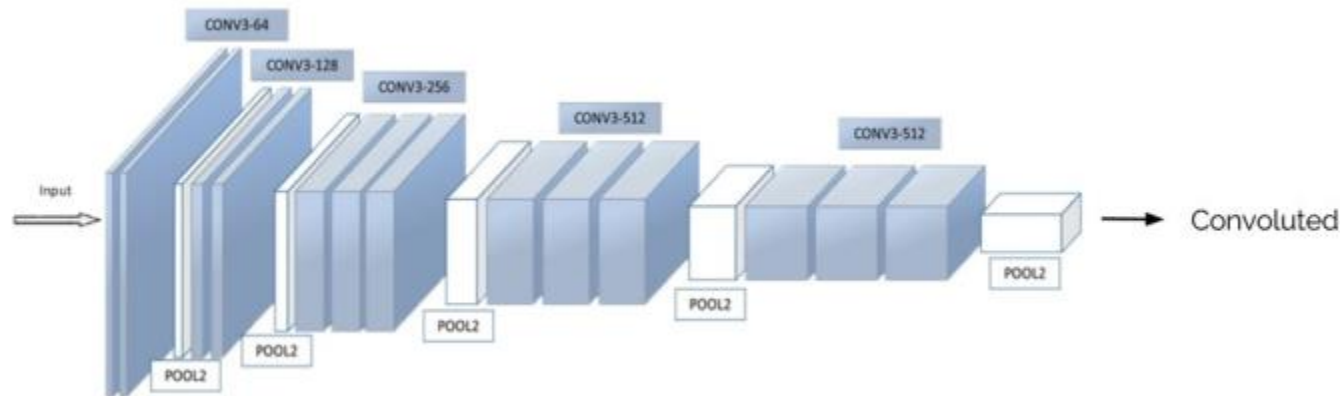


```
# import necessary modules
import time
import matplotlib.pyplot as plt
import numpy as np
% matplotlib inline
np.random.seed(2017)
from keras.applications.vgg19 import VGG19
from keras.applications.vgg19 import preprocess_input, decode_predictions
from keras.preprocessing import image
from keras.models import Model
import cv2
```

# Begriffe – Transfer Learning

## ■ Transfer Learning

```
# load pre-trained model
model = VGG19(weights='imagenet', include_top=True)
# display model layers
model.summary()
```



Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2
block3_conv4 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv3
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv4
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2
block4_conv4 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv3
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv4
block5_conv1 (Convolution2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv1
block5_conv3 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv2
block5_conv4 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv3
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv4
flatten (Flatten)	(None, 25088)	0	block5_pool[0][0]
fc1 (Dense)	(None, 4096)	102764544	flatten[0][0]
fc2 (Dense)	(None, 4096)	16781312	fc1[0][0]
predictions (Dense)	(None, 1000)	4097000	fc2[0][0]

Total params: 143,667,240  
 Trainable params: 143,667,240  
 Non-trainable params: 0

# Begriffe – Transfer Learning

## ■ Transfer Learning

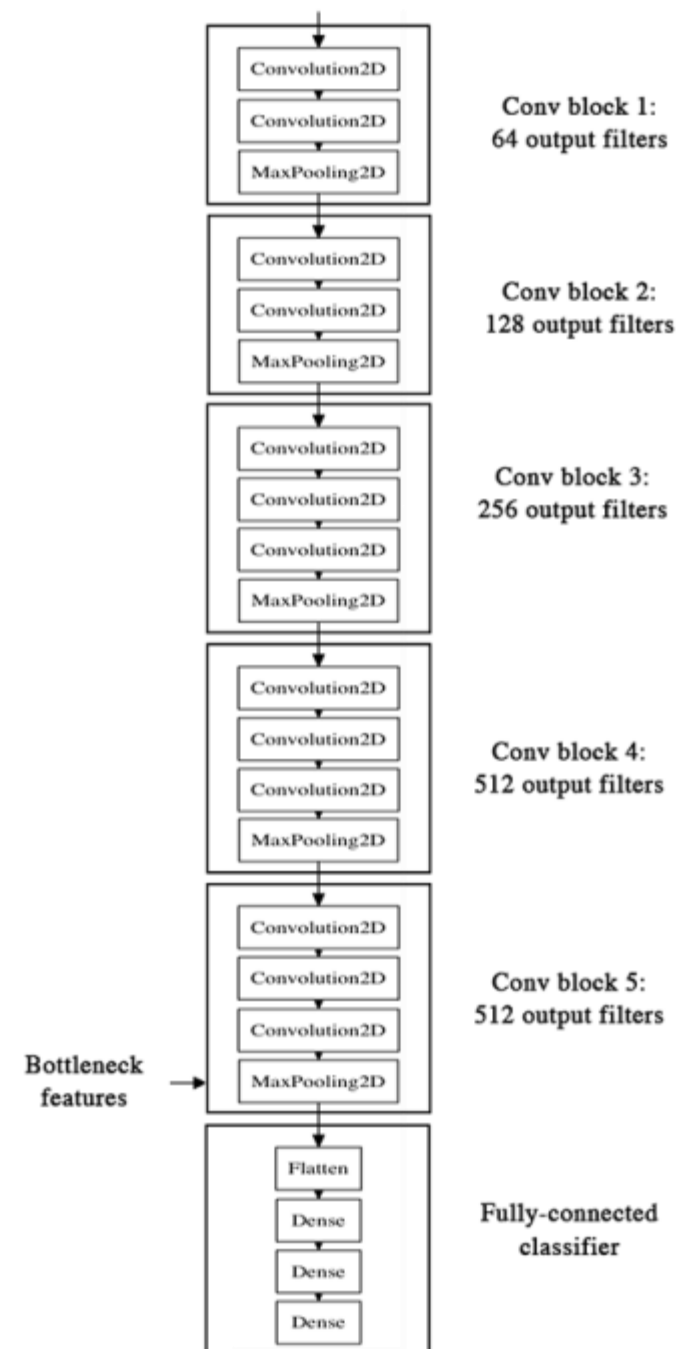
1. Erstelle Output vom pretrained model
2. Trainiere Fully-connected layer mit diesem output

*oder*

1. Trainiere gesamtes Netz mit "Freezing" der Layer im CNN

```
model.layers.pop()
model.outputs = [model.layers[-1].output]
model.layers[-1].outbound_nodes = []
model.add(Dense(num_class, activation='softmax'))
```

```
for layer in model.layers[:10]:
    layer.trainable = False
```

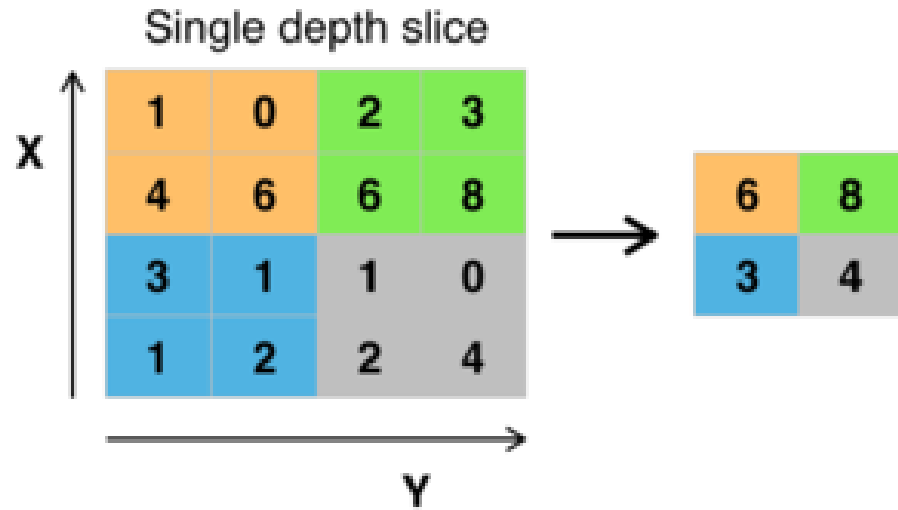






# Begriffe – Max Pooling

- Max Pooling (<http://ufldl.stanford.edu/tutorial/supervised/Pooling/>)



Hinton über Max Pooling:

*"The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster. If the pools do not overlap, pooling loses valuable information about **where** things are. ..."*



# Allgemeines

Cheatsheets für neuronale netze

<https://becominghuman.ai/cheat-sheets-for-ai-neural-networks-machine-learning-deep-learning-big-data-678c51b4b463>

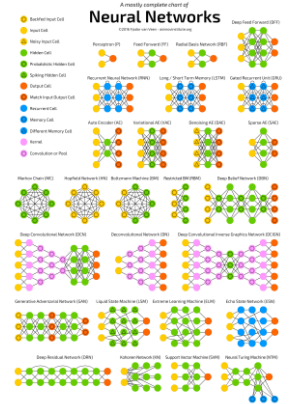
Link zu Tensorflow

<https://becominghuman.ai/an-introduction-to-tensorflow-f4f31e3ea1c0> Great book



# Netzwerkarchitekturen

- MLP
- AutoEncoder
- CNN (<http://scs.ryerson.ca/~aharley/vis/conv/>)
- RNN
- LSTM
- GAN
- Capsule
- ...



<http://www.asimovinstitute.org/neural-network-zoo/>














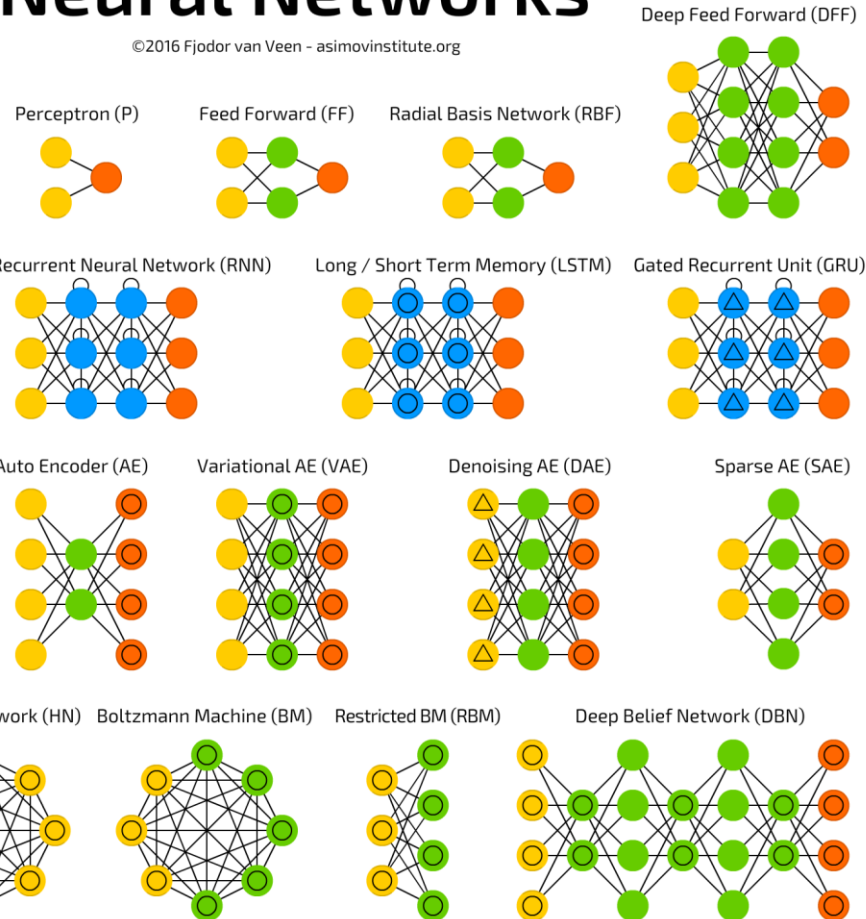
# Netzwerkarchitekturen

A mostly complete chart of

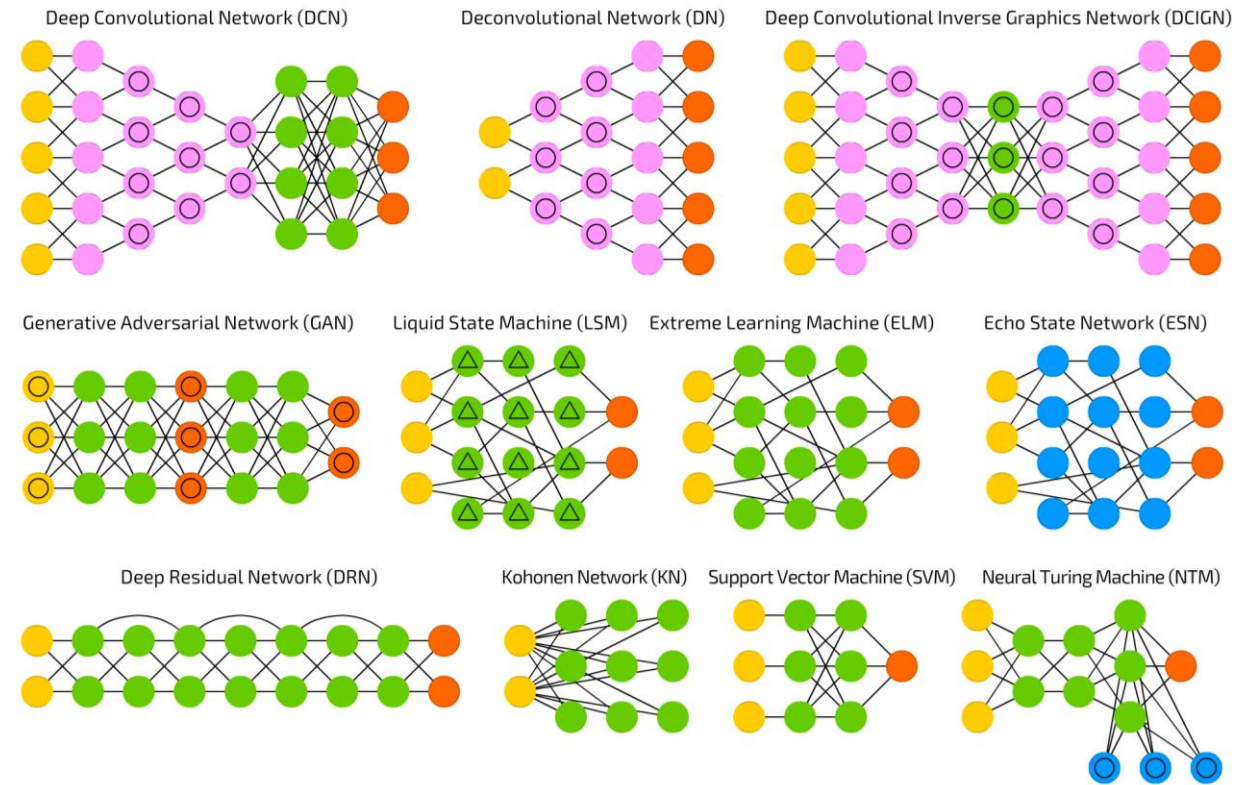
## Neural Networks

©2016 Fjodor van Veen - asimovinstitute.org

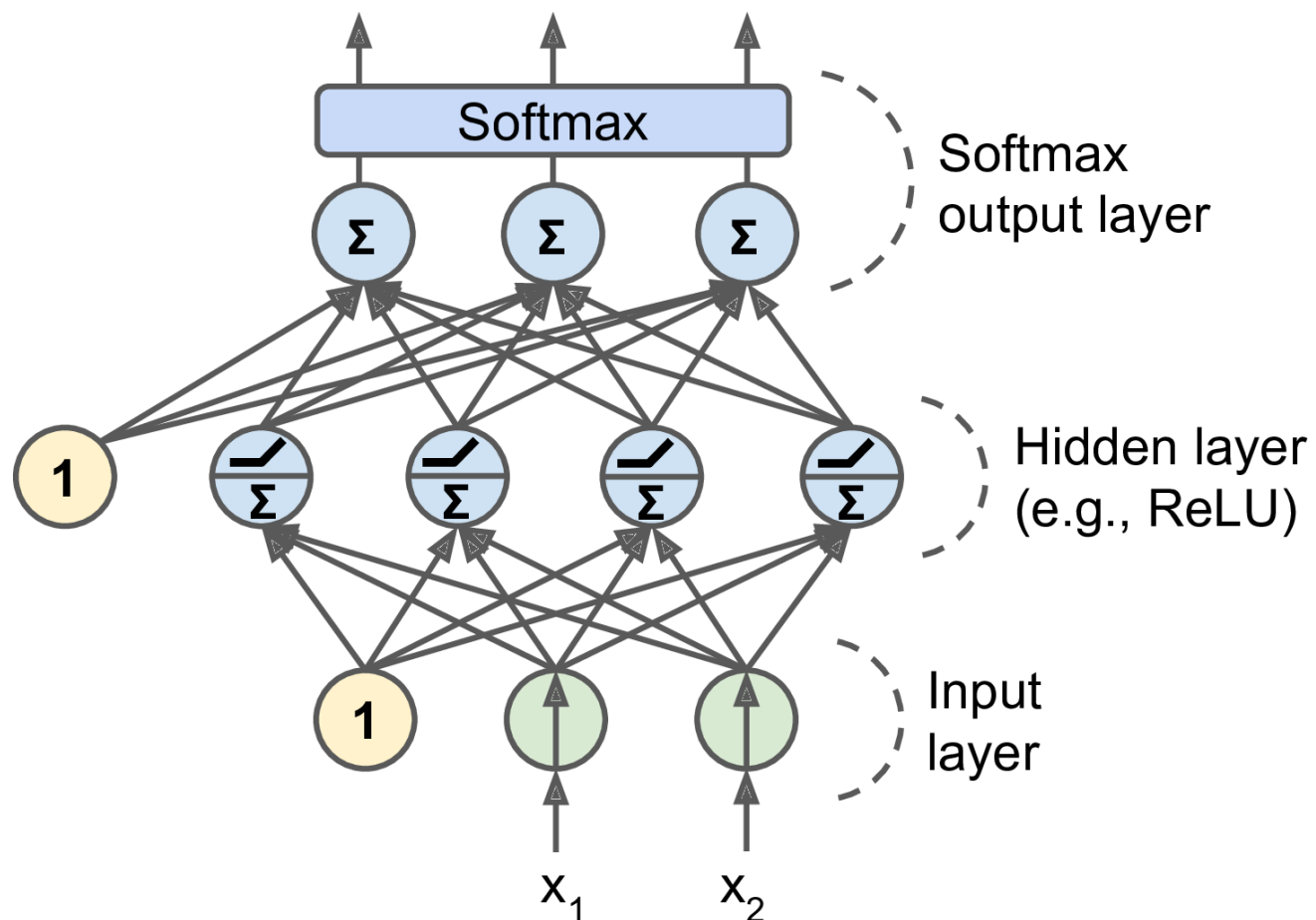
-  Backfed Input Cell
-  Input Cell
-  Noisy Input Cell
-  Hidden Cell
-  Probabilistic Hidden Cell
-  Spiking Hidden Cell
-  Output Cell
-  Match Input Output Cell
-  Recurrent Cell
-  Memory Cell
-  Different Memory Cell
-  Kernel
-  Convolution or Pool



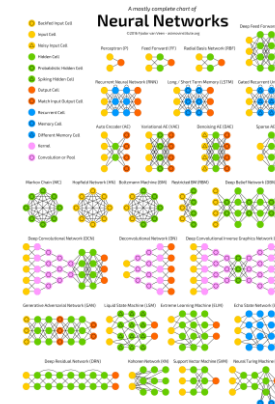
<http://www.asimovinstitute.org/neural-network-zoo/>



# Netzwerkarchitekturen - MLP



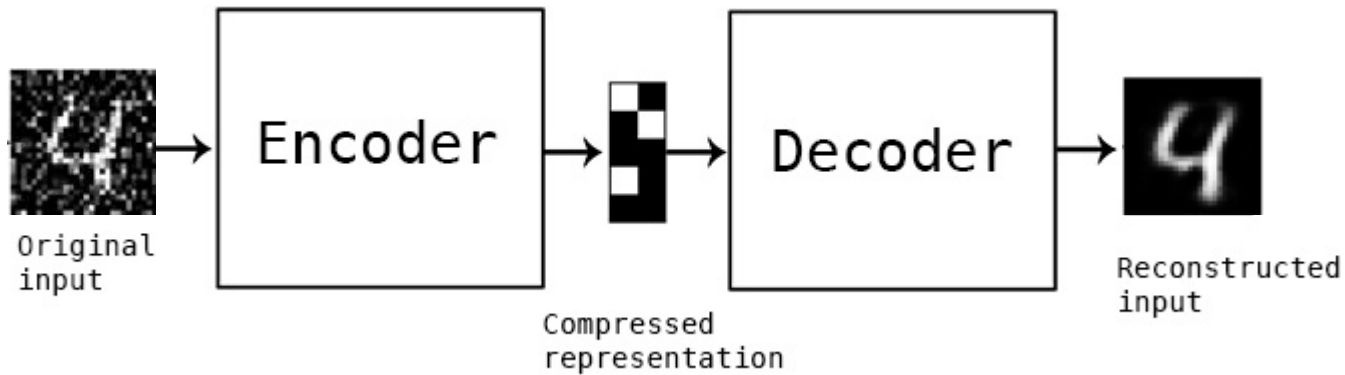
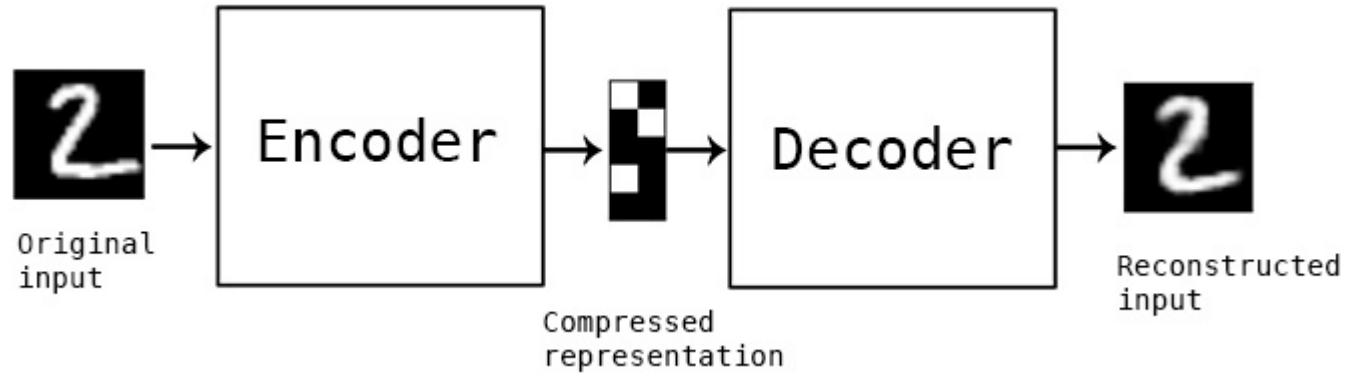
# Playground







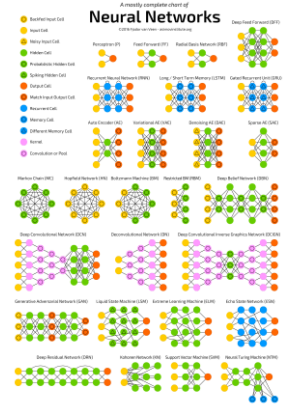
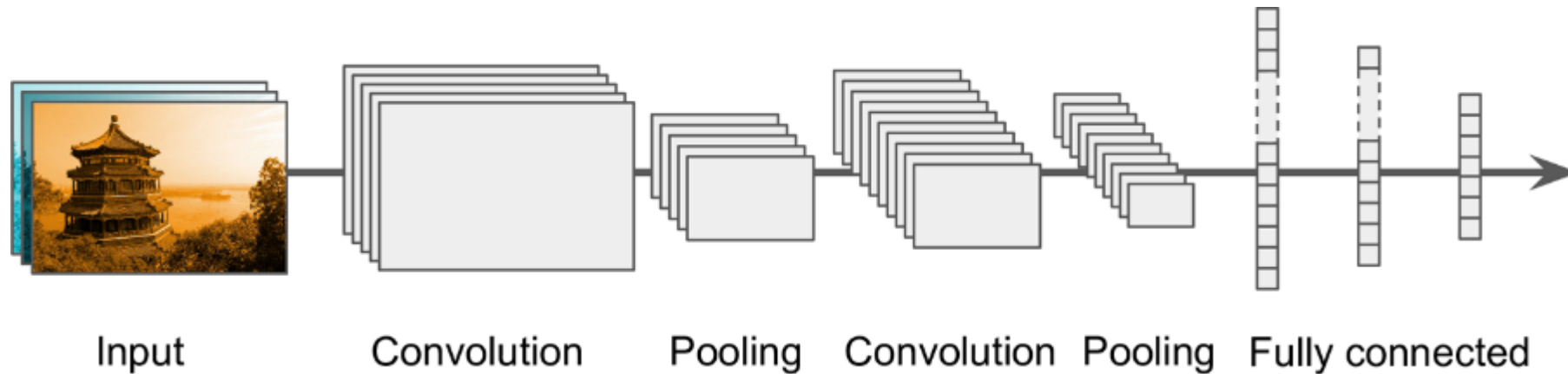
# Netzwerkarchitekturen - Autoencoder





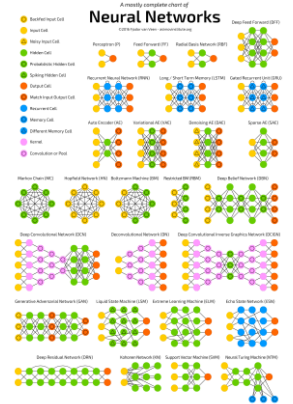
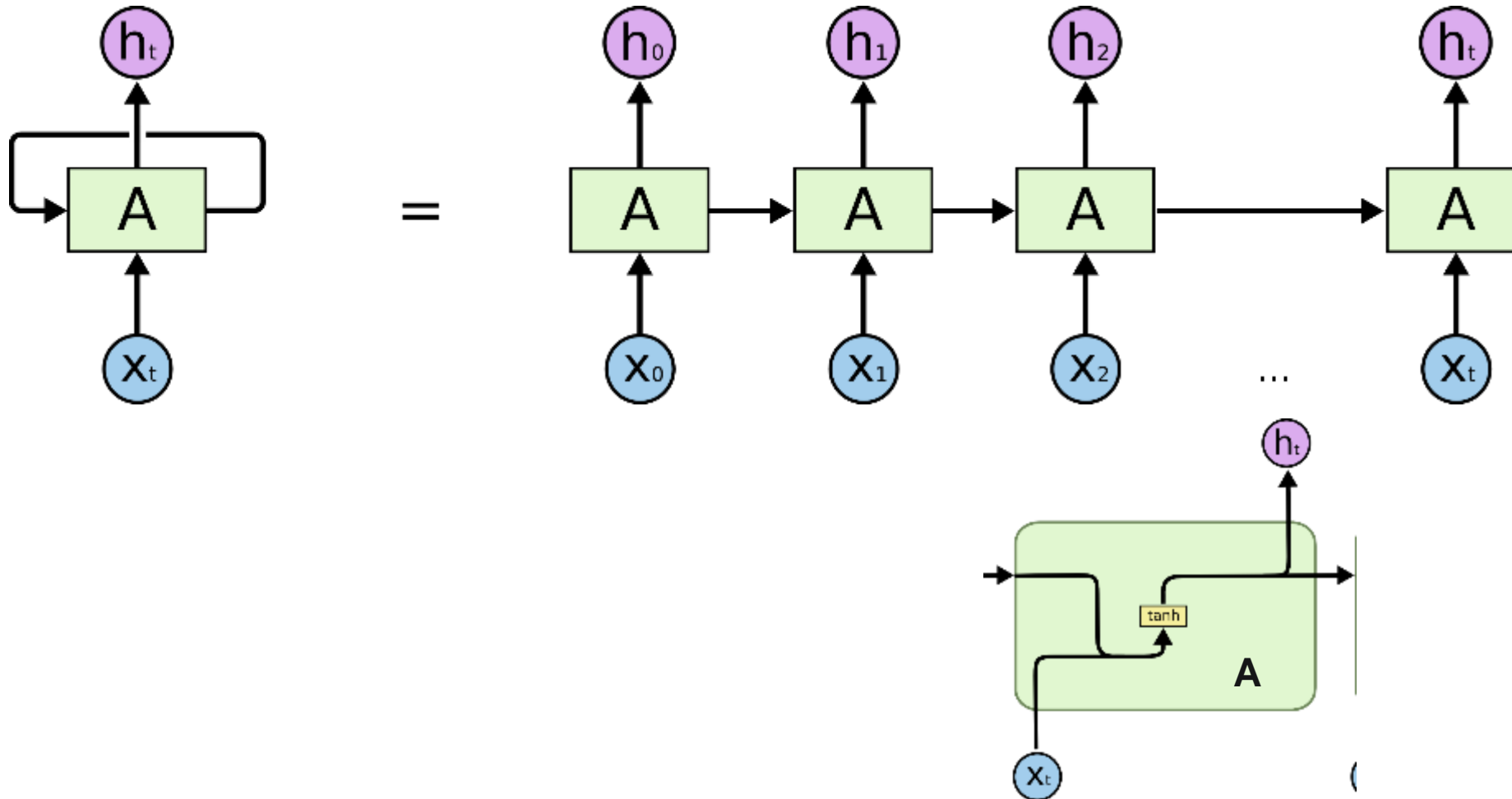
# Netzwerkarchitekturen - CNN

- CNN (<http://scs.ryerson.ca/~aharley/vis/conv/>)



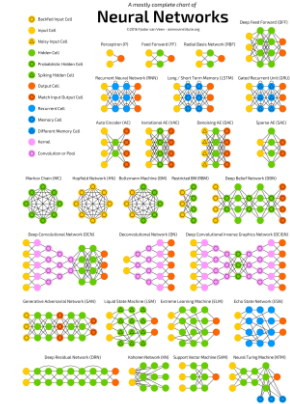
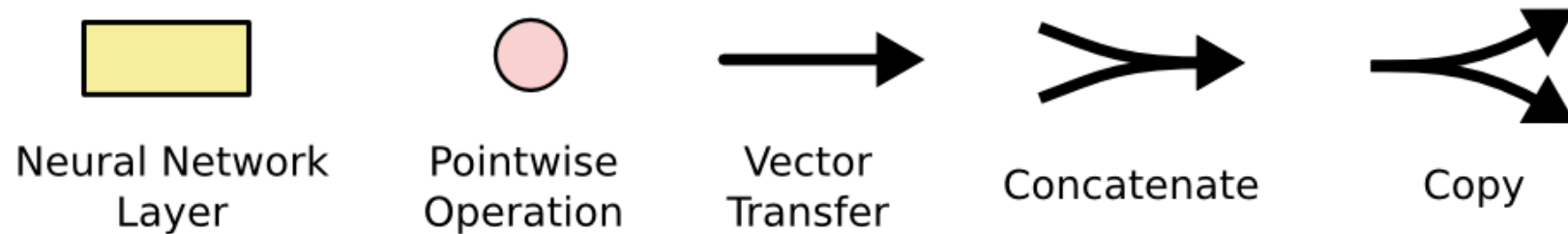
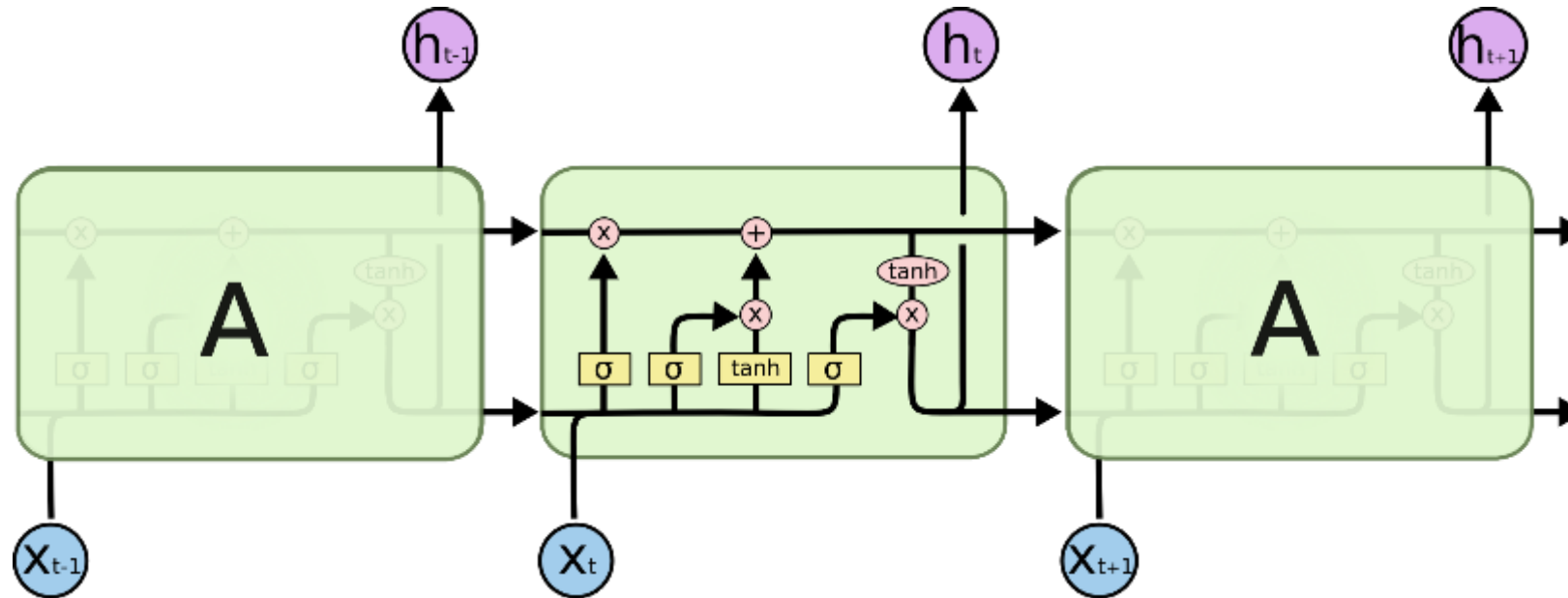


# Netzwerkarchitekturen - RNN





# Netzwerkarchitekturen - LSTM



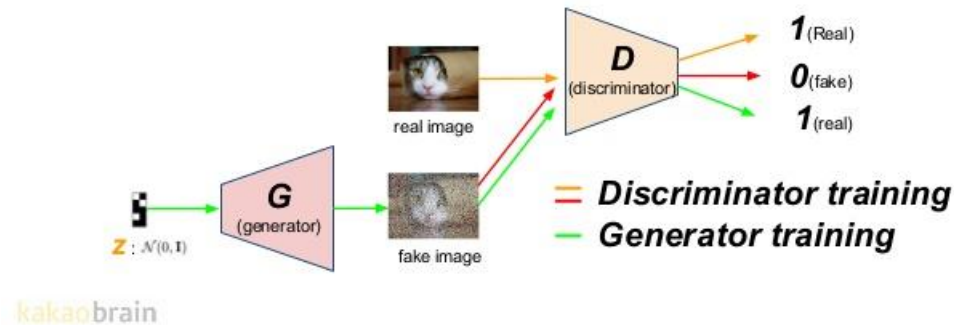


# Netzwerkarchitekturen - GAN

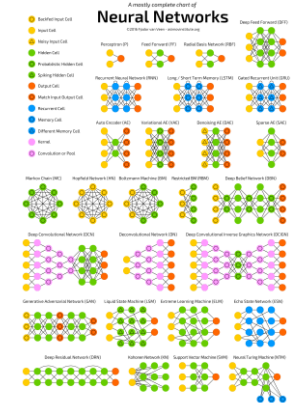
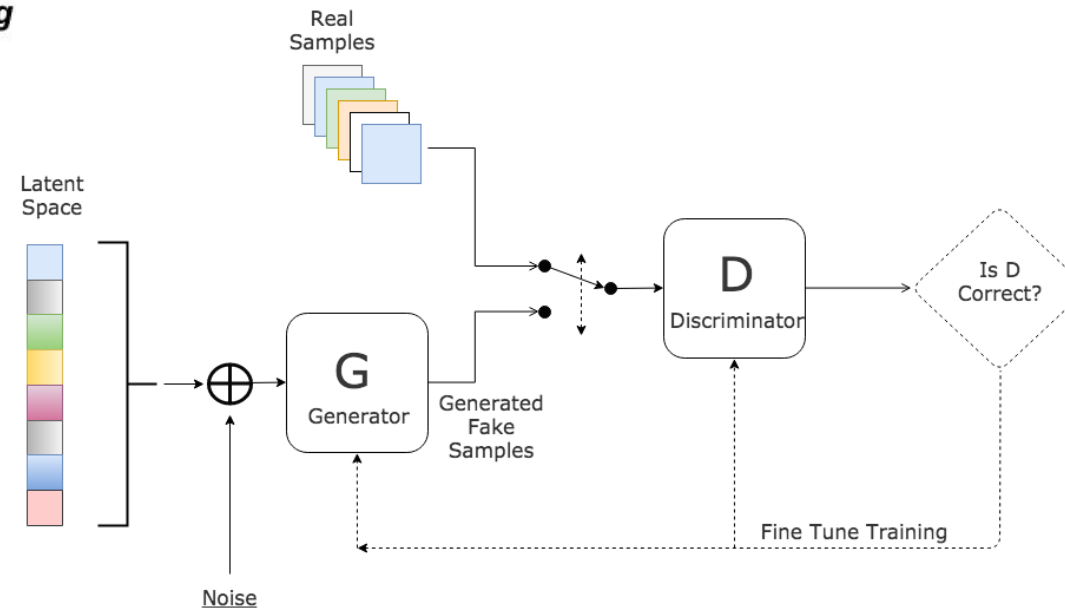
GAN

## Generative Adversarial Networks - GAN

- Alternate training
- [https://github.com/buniburisuri/sugartensor/blob/master/sugartensor/example/mnist\\_gan.py](https://github.com/buniburisuri/sugartensor/blob/master/sugartensor/example/mnist_gan.py)

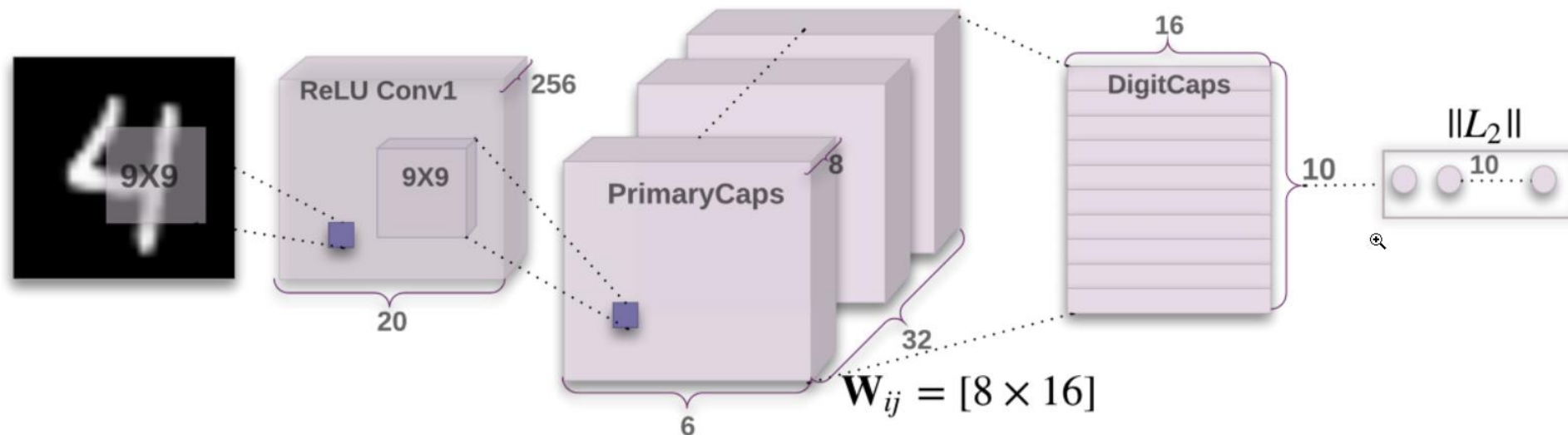
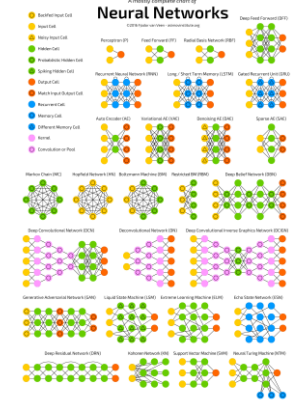
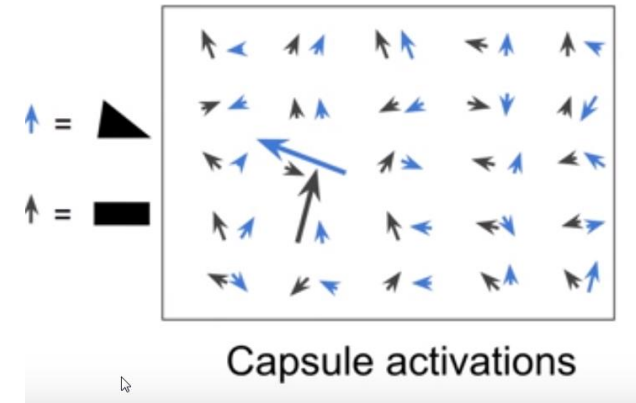
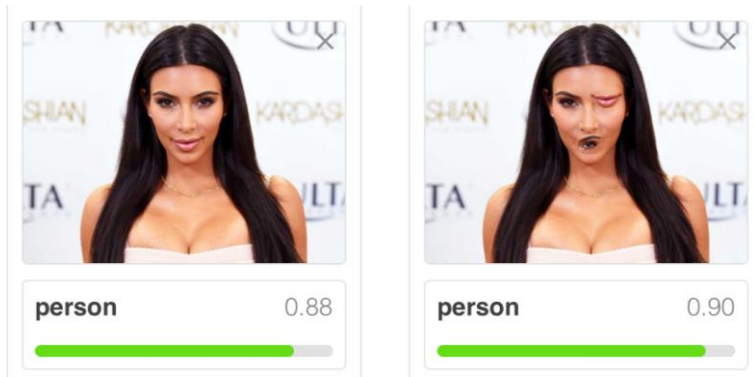


## Generative Adversarial Network





# Netzwerkarchitekturen – Capsule, CapsNet







# Libraries

	Caffe	Torch	Theano	TensorFlow
<i>language</i>	Python, C++	Lua	Python	Python, Java, C, Go
<i>pre-trained models</i>	Model Zoo	ModelZoo	Lasagne	Inception, others
<i>parallel GPUs: data</i>	Yes	Yes	Yes	Yes
<i>parallel GPUs: model</i>		Yes		Yes
<i>source code</i>	Readable	Readable		
<i>for RNNs</i>			Good	Best
<i>high-level APIs</i>			Keras	Keras, TFLearn



# Deep Learning Machines

## In der Cloud

- Google Cloud Computing
- Amazon (AWS)
- ...

## GPU-Maschinen

- NVIDIA
- Tutorials zum Selbstbau



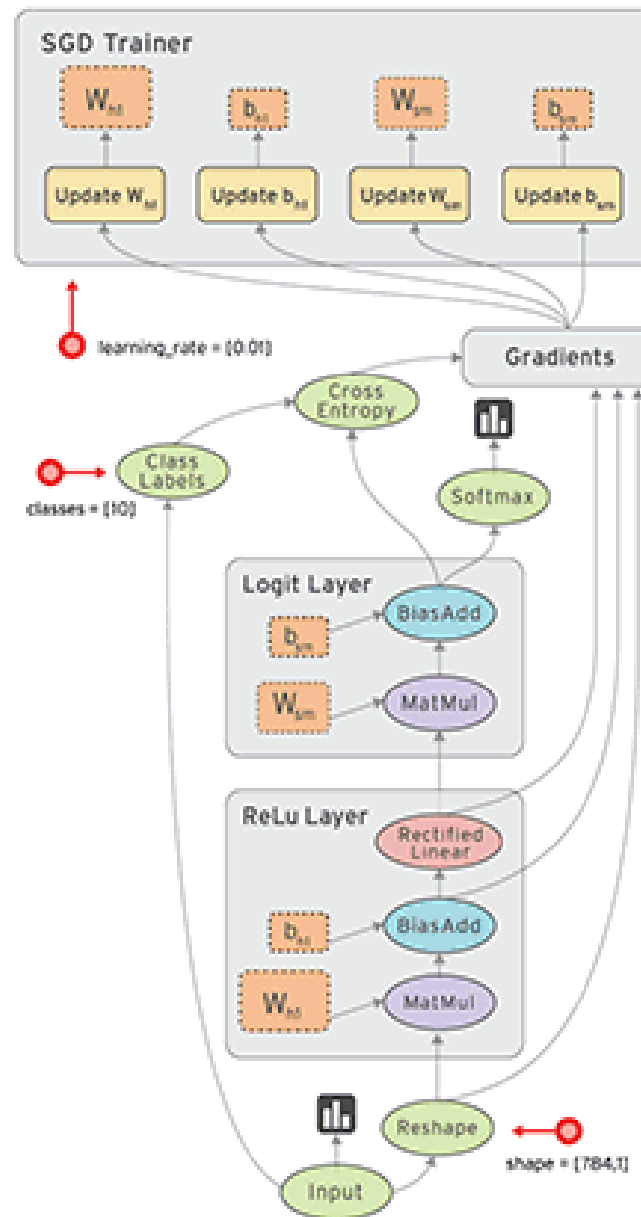
# Einführung Tensorflow

## ■ Tensor

Rank	Math entity	Shape	Dimension number	Example
0	Scalar (magnitude only)	$[]$	0-D	A 0-D tensor. A scalar.
1	Vector (magnitude and direction)	$[D0]$	1-D	A 1-D tensor with shape $[5]$ .
2	Matrix (table of numbers)	$[D0, D1]$	2-D	A 2-D tensor with shape $[3, 4]$ .
3	3-Tensor (cube of numbers)	$[D0, D1, D2]$	3-D	A 3-D tensor with shape $[1, 4, 3]$ .
n	n-Tensor (you get the idea)	$[D0, D1, \dots D_{n-1}]$	n-D	A tensor with shape $[D0, D1, \dots D_{n-1}]$ .

# Einführung Tensorflow

## ■ Flow





# Einführung Tensorflow

## Beispiele

1. Einfacher Graph mit TensorFlow



2. Neuron mit TensorFlow



3. [DeepNet – TensorFlow](#)

4. [DeepNet – Keras](#)

5. [Convolutional Net – TensorFlow](#)

6. [Convolutional Net – Keras](#)