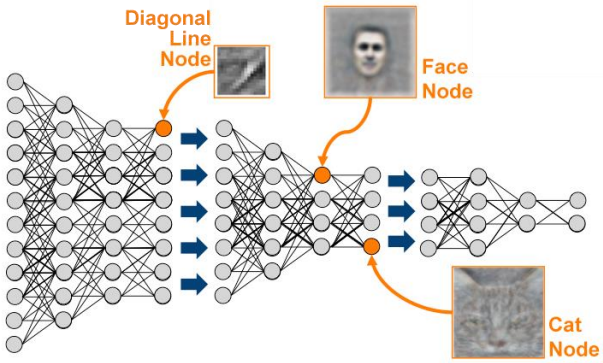
Hochschule Offenburg
University of Applied Sciences

Künstliche Intelligenz
7. Neuronale Netzwerke

Prof. Dr. Klaus Dorer



Übersicht

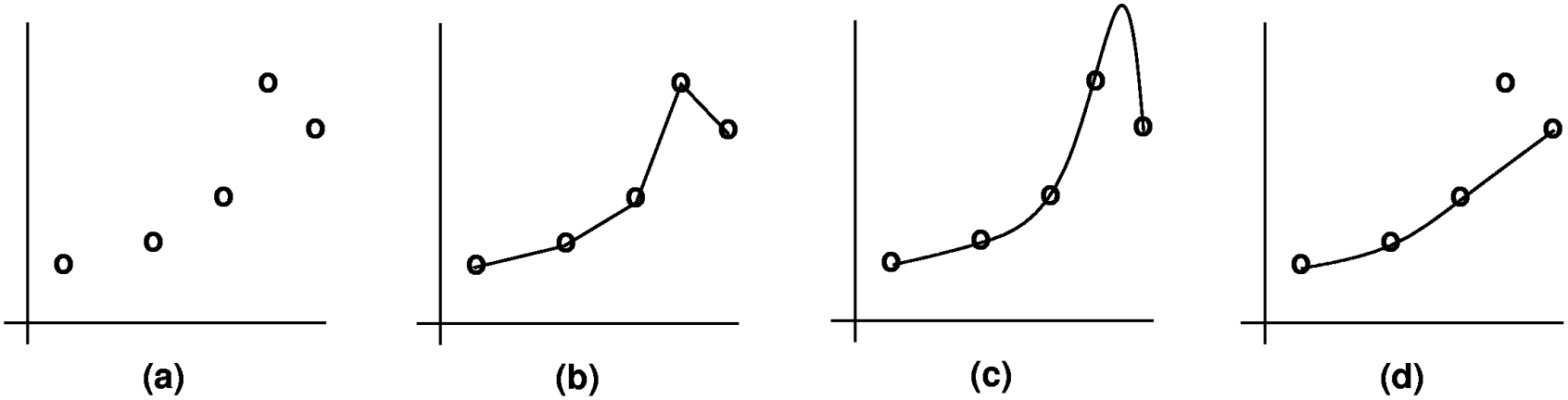
- Induktives Lernen
- Neuronale Netzwerke
  - Menschliches Gehirn
  - Feedforward Netzwerke
    - Perceptron
    - Backpropagation
  - Recurrent Networks
  - Deep Learning
- Ziele
  - Funktionsweise und Einschränkungen von neuronalen Netzwerken verstehen
  - Anwendungsmöglichkeiten einschätzen können

Quellen

- Russel, Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, ISBN 0137903952, 2002.
- Jürgen Weiprecht, Neuronen, Modelle, Künstliche Neuronale Netze, 2004
- www.wikipedia.de
- Java Version des SNNS Stuttgarter Neural Network Simulator (Andreas Zell u.a.) http://www.ra.cs.uni-tuebingen.de/downloads/JavaNNS/
- Wolfram Lippe, Einführung in Neuronale Netze, Uni Osnabrück
- Human Brain Project https://indico.desy.de/getFile.py/access?contribId=8&resId=0&materialId=slides&confId=6264
- https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab

Induktives Lernen

- Ein Beispiel ist ein Paar (x, f(x)), x – input, f(x) – output
- Induktion generiert aus einer Menge von Beispielen eine Funktion h (Hypothese), die f approximiert
- Es gibt meist viele mögliche Hypothesen
- Die Bevorzugung einer nennt man Bias
  - Beispiel: Ockham's razor : wähle die 'einfachste' Hypothese
- Ziel: für unbekannte Inputs möglichst gut Outputs vorhersagen (Generalisierung)

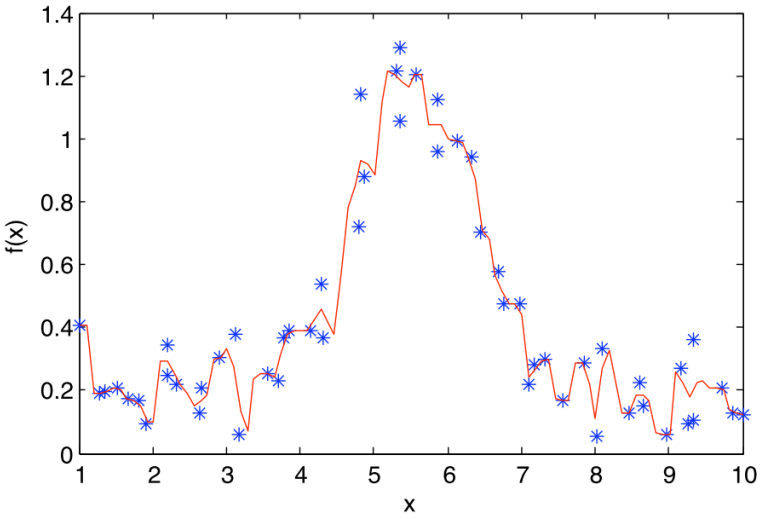


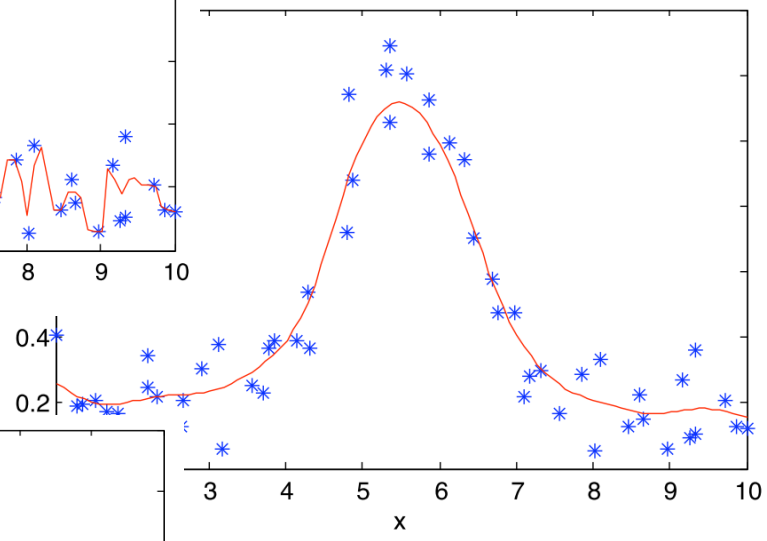
Supervised Learning

- Trainingsset mit Input und gewünschtem Output bekannt
- Kategorischer Output
  - Klassifikation
  - {ja, nein} oder {Frühling, Sommer, Herbst, Winter}
  - Fehlerkriterium: missclassification rate (z.B. 3%)
- Quantitativer Output
  - Regression
  - 54.5 km/h
  - Fehlerkriterium: residual sum of squares RSS $\sum_{i=1}^{n} (y_i - h(\vec{x}_i))^2$

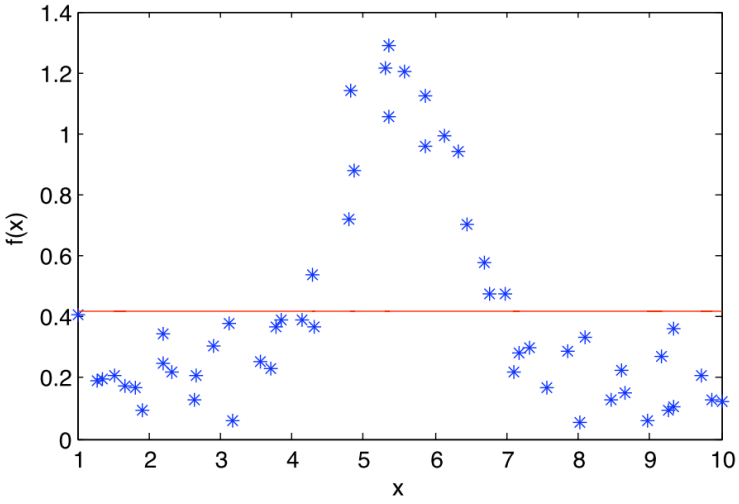
Vorhersagequalität

- Overfitting
  - auch die Messfehler wurden gelernt

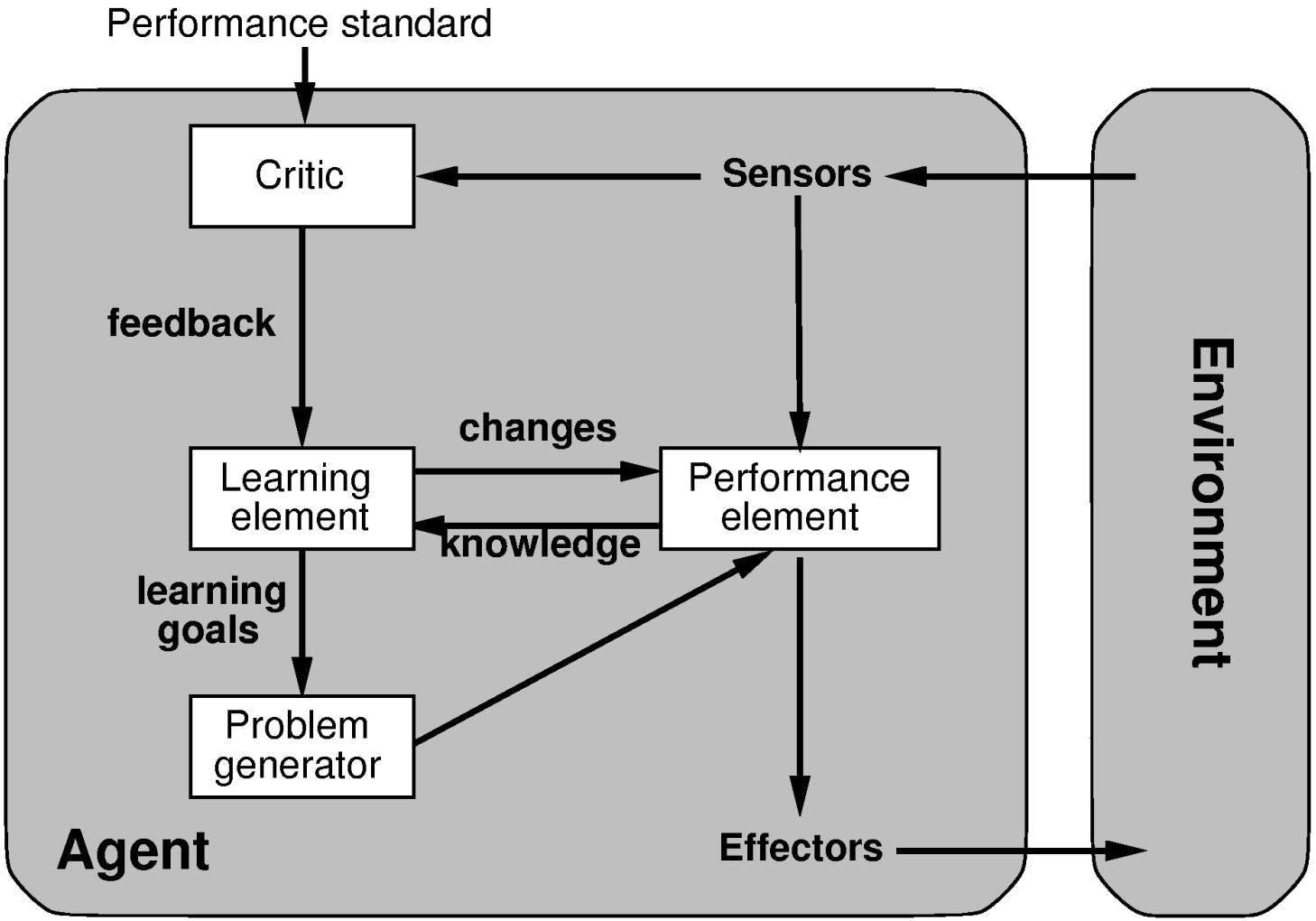




- Underfitting
  - Funktion generalisiert nicht



Modell eines lernenden Agenten



Performance Element

- Aktionsauswahl basierend auf Wahrnehmungen und Wissen
- Was wir bisher als Agent bezeichnet haben
- Mögliche Komponenten
  - Direkte Abbildung von Wahrnehmungen zu Aktionen
  - Modul, das aus Wahrnehmungen Eigenschaften der Welt inferiert
  - Information über die Entwicklung der Umgebung
  - Information über die Effekte von Aktionen
  - Information über den Nutzen von Zuständen
  - Aktion-Wert Information über den Nutzen von Aktionen in bestimmten Zuständen
  - Ziele, die Klassen von Zuständen beschreiben, die den Nutzen des Agenten maximieren

Beispiele Performance Element

- Lookup Table
  - Vollständige Tabelle, die für jede Situation die optimale Aktion enthält
  - Schnellste aller Möglichkeiten, optimal, vollständig
  - Größe der Tabelle ist prohibitiv
- Regelsystem
  - Menge von (disjunkten) wenn-dann Regeln
  - Generalisierung über Mengen von Zuständen möglich
  - Es ist nicht trivial, disjunkte Regelwerke zu erstellen
- Entscheidungsbäume
  - Entscheidungslogik wird durch einen Baum repräsentiert, dessen Knoten Attribute und Kanten Attributwerte sind. Die Blätter repräsentieren die Entscheidung.
  - ‘Regeln’ sind automatisch disjunkt
- Neuronale Netzwerke
- …

Critic

- Meldet dem Agenten, wie gut er performt
- Der Performanz-Standard muss konzeptionell außerhalb des Agenten sein damit der Agent ihn nicht anpassen kann (Fuchs und Weintrauben)
- Bestimmt die Art des Lernens
  - supervised : input und output werden wahrgenommen
  - reinforcement : Agent erhält Bewertung verzögert
  - unsupervised : kein Hinweis auf den korrekten Output
- Beispiele
  - Fitness Funktion bei genetischen Algorithmen
  - Zieloutput – Aktueller Output bei neuronalen Netzwerken
  - Reward Funktion bei reinforcement learning

Learning Element

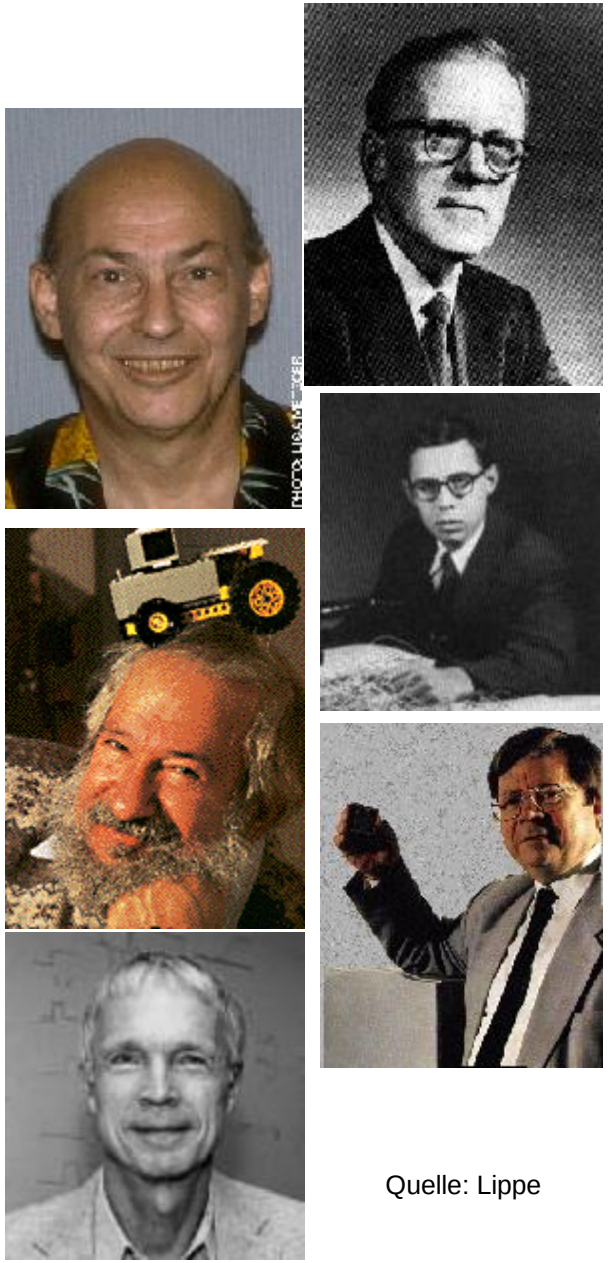
- Zuständig für Verbesserungen
- Aus Wissen über das Performance Element und Rückmeldungen des Critic Elements generiert es Änderungen des Performance Elements
- Design hängt stark vom Design des Performance Elements ab
  - am Anfang steht die Entscheidung, wie mein Agent aussehen muss um die zu lernende Aufgabe durchzuführen
  - dann erst wie das Lernen durchgeführt wird
- Beispiele
  - Genetischer Algorithmus
  - Decision Tree Induktion durch Information Gain
  - Backpropagation

Problem Generator

- Zuständig für die Exploration des Problemraums
- Das Performance Element würde immer die beste bekannte Aktion auswählen
- Es könnte aber bessere, unbekannte Aktionen geben
- Tradeoff zwischen Exploration und Exploitation

Geschichte

- 1943 Mc Culloch – Pitts
  - Modell eines Neurons
- 1949 Donald Hebb
  - Lernregel $W_{j,i} = W_{j,i} + \alpha \cdot o_i \cdot a_j$
- 1951 Minsky
  - Snark Neurocomputer
- 1957 Rosenblatt
  - Perceptron
- 1969 Minsky und Papert
  - Problem der linearen Separierbarkeit
- 1972 Kohonen
  - Linear assoziierende Netzwerke
- 1982 Hopfield
  - Autoassoziative Netzwerke
- 1986 Rummelhart und McClelland
  - Backpropagation
- 2001 Jaeger
  - Echo State Netzwerke
- 2012 LeCun, Hinton, Ng, Schmidhuber, …
  - Deep learning Netzwerke



Quelle: Lippe

Menschliches Gehirn - Computer

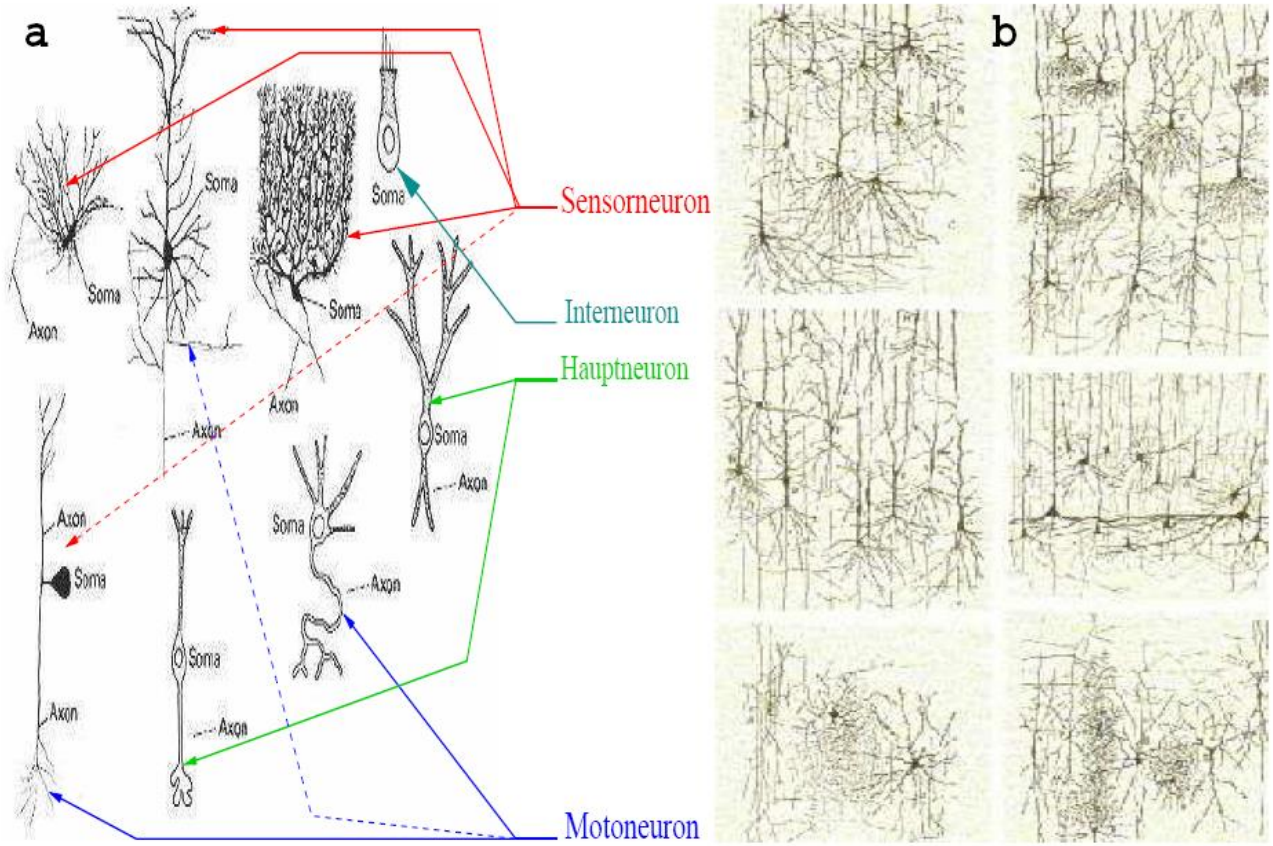
	Mensch	Computer
		
Recheneinheiten	10^{13} Neuronen	1 CPU (10^{6} gates)
Speichereinheiten	10^{13} Neuronen 10^{16} Synapsen	10^{10} Bit RAM 10^{13} Bit Disk
Zykluszeit	10^{-3} sek	10^{-8} sek
Bandbreite	10^{14} Bit/sek	10^{10} Bit/sek
Neuronen updates/sec	10^{5}	10^{14}

Menschliches Gehirn - Computer

	Mensch	Computer
		
Arbeitsweise	Massiv parallel Assoziativ	Extrem kurze Zykluszeit Adressbasiert
Fehlertoleranz	Nervenzellen sterben ständig ohne merklichen Performanzverlust	Defektes Memory Bit kann Applikation komplett unbrauchbar machen
Lebenszeit	~80 Jahre Keine Reboots nötig	~10 Jahre Reboot täglich – 1x/Monat
Programmierung	Induktives Lernen	Programmiersprachen

Menschliches Gehirn

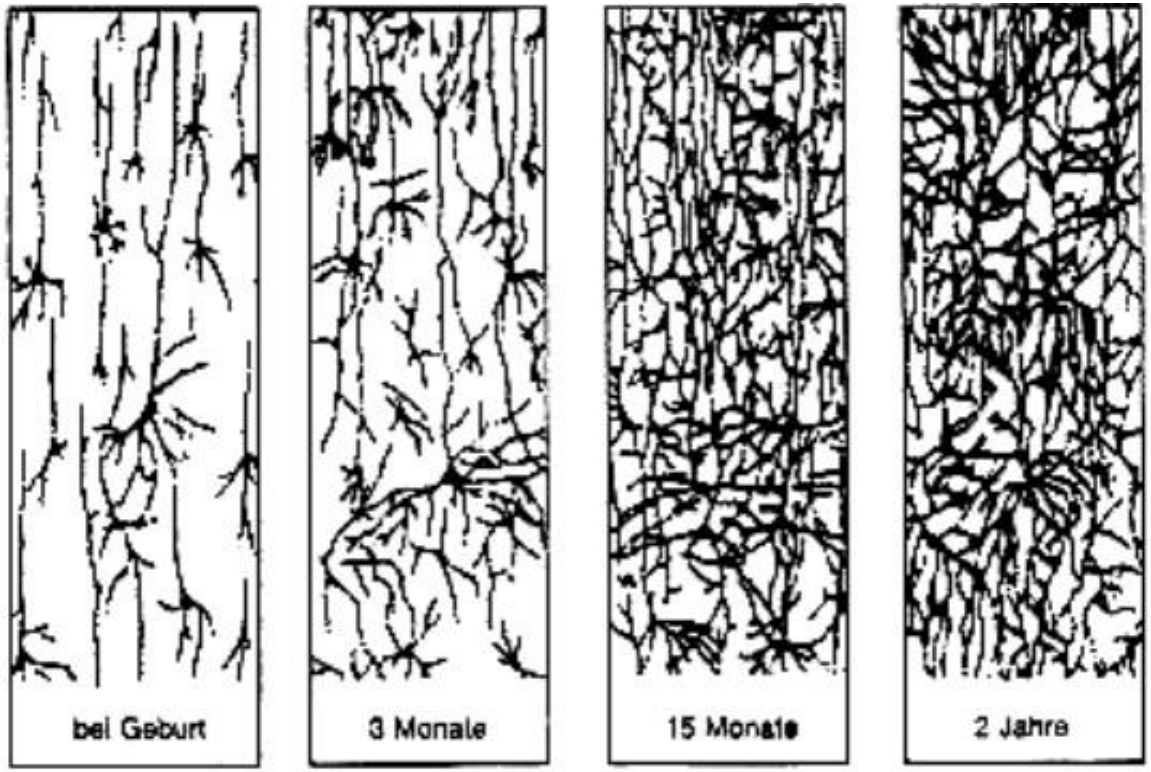
- Gehirn ist ein Netzwerk von Neuronen
- Es gibt viele verschiedene Typen von Neuronen



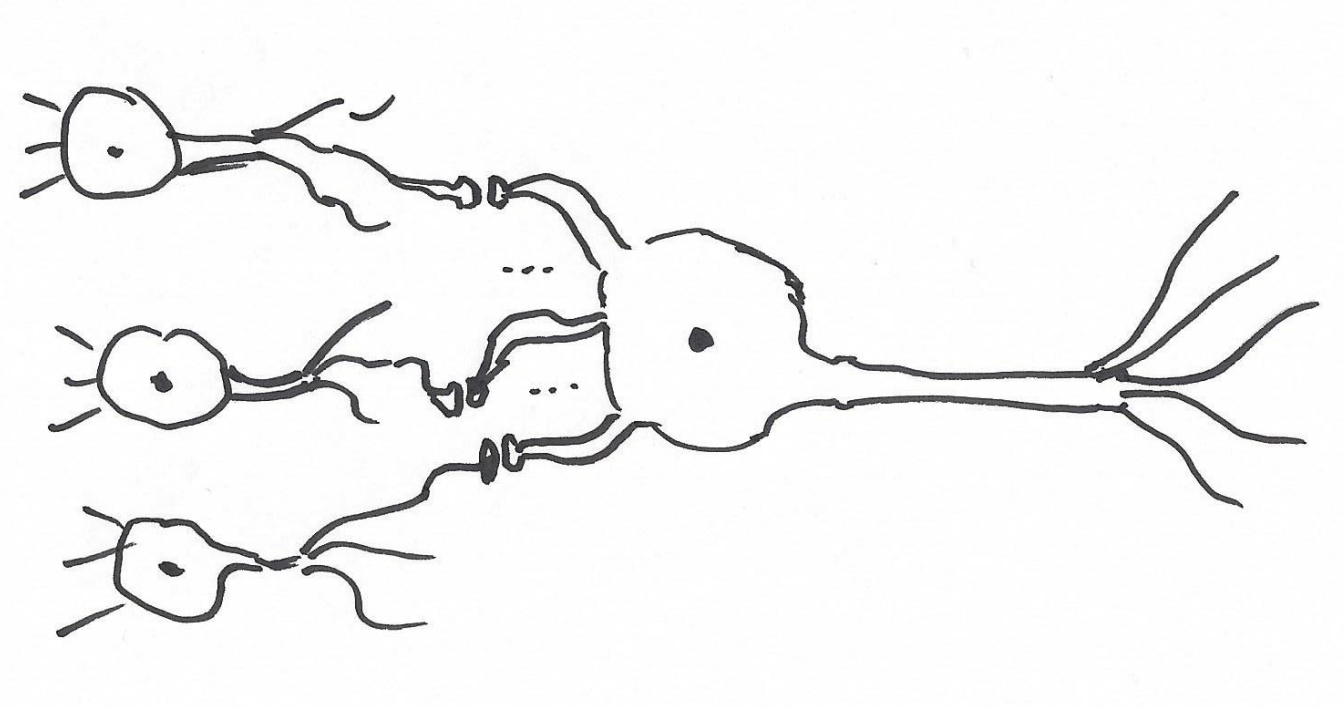
Quelle: Weiprecht

Menschliches Gehirn

- Lernen erfolgt durch
  - Bildung neuer Verbindungen zwischen Nervenzellen
  - Verstärkung von Verbindungen

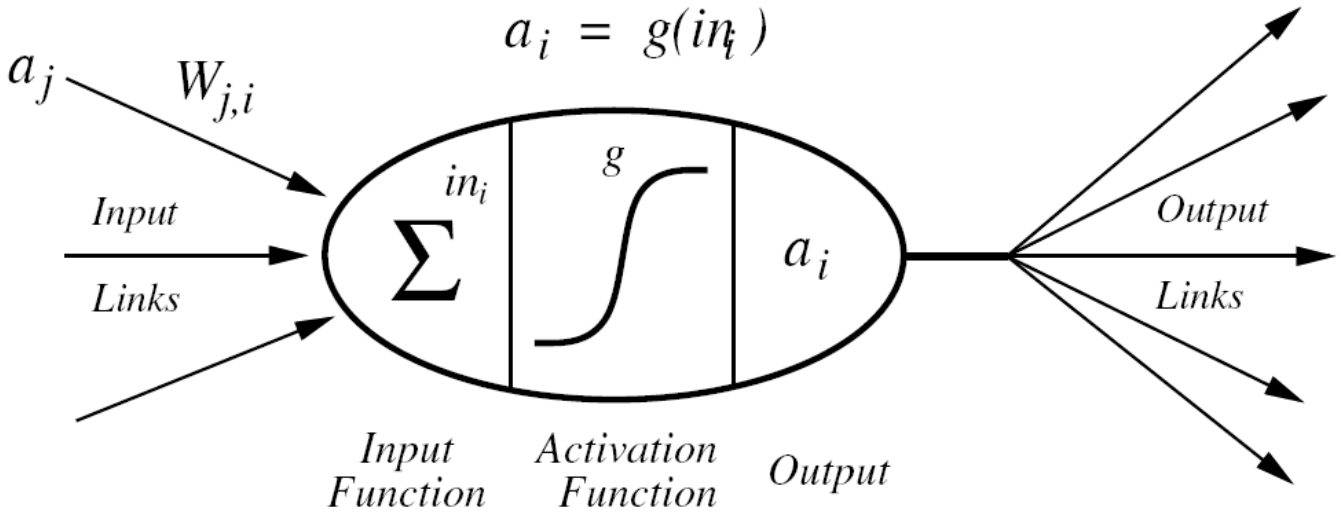


Menschliches Gehirn



Modell einer Nervenzelle

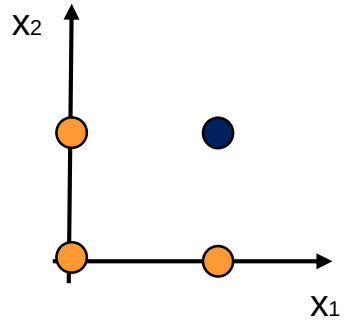
- Inputs liefern Eingangssignal
- Stärke hängt von Gewichten $W_{j,i}$ ab
- Output ist Ergebnis einer Aktivierungs-funktion der gewichteten Inputs
- Wird als Output an andere Neuronen weitergegeben

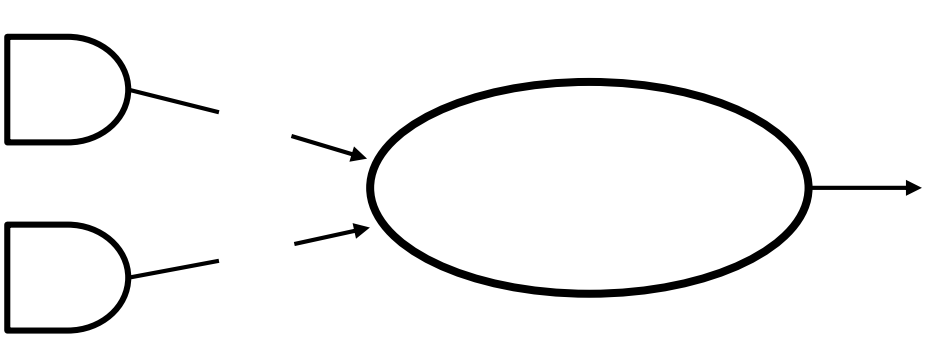


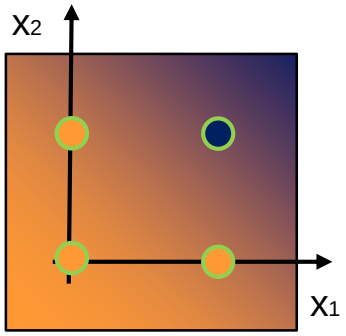
Quelle: Russel, Norvig

Modell einer Nervenzelle
Beispiel: And Funktion

x_1	x_2	y_1
0	0	0
0	1	0
1	0	0
1	1	1



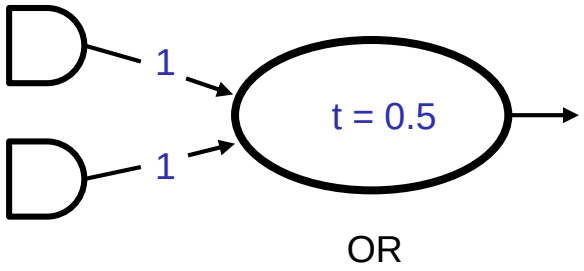




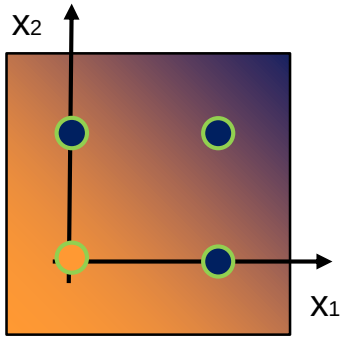
$$o_j = Step_t(\sum_i w_{i,j} x_i) = Step_t(Wx)$$

Modell einer Nervenzelle
Or und Not Funktion

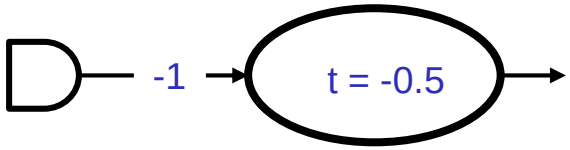
x_1	x_2	y_1
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**



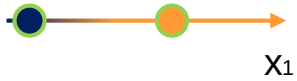
OR



x_1	y_1
0	**1**
1	**0**

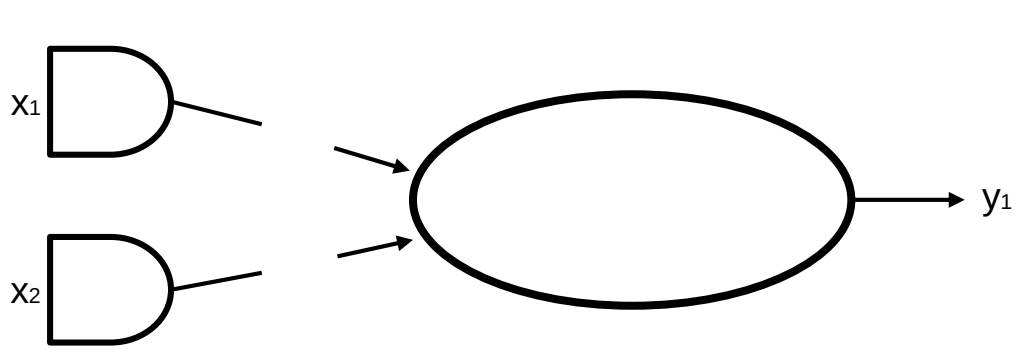


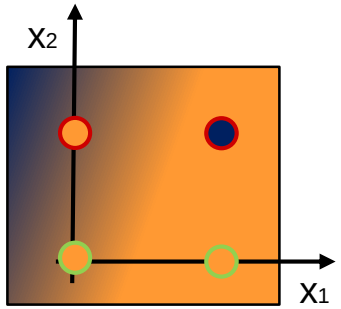
NOT



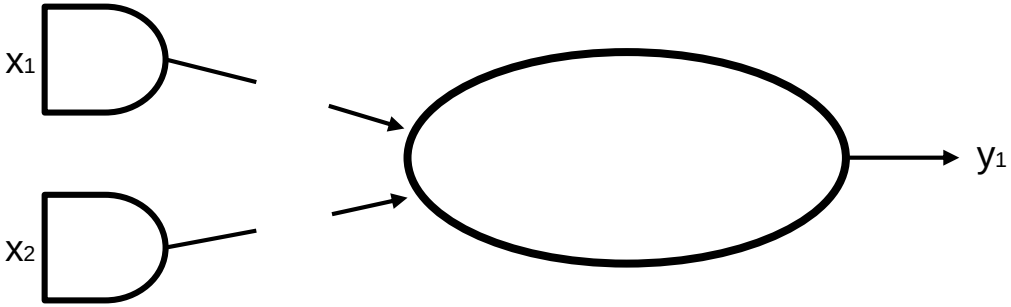
Modell einer Nervenzelle

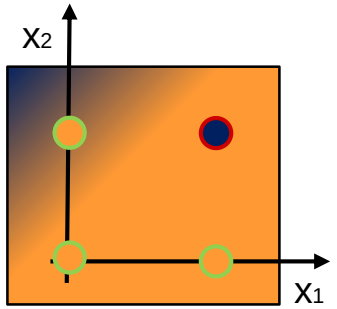
Passende Werte finden





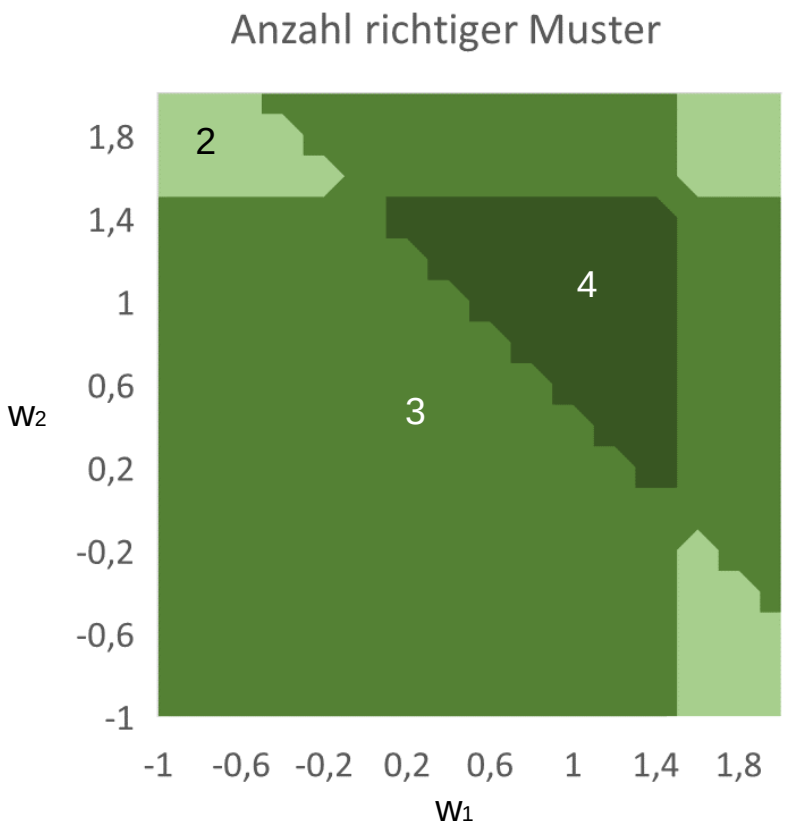
AND



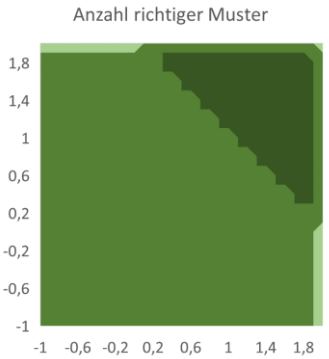


Modell einer Nervenzelle
Passende Werte finden

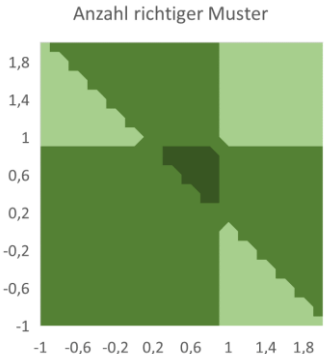
t=1.5



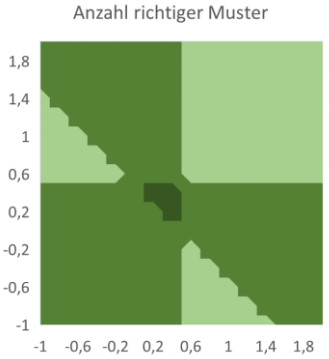
t=2



t=1

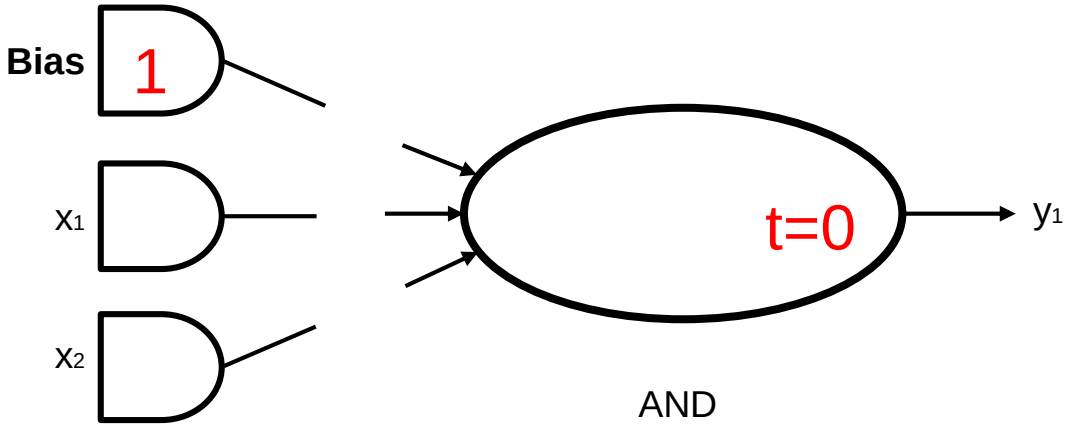


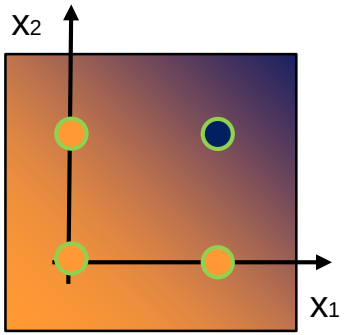
t=0.5



Modell einer Nervenzelle
Passende Werte finden

- Bias





$$o_j = Step_0(\sum_{i+1} w_{i,j} x_i)$$

Modell einer Nervenzelle
Passende Werte finden

- Gewichte w seien -1 und 1
- Fehlerfunktion (loss function)
  - Betragsfehler (L1 Norm)

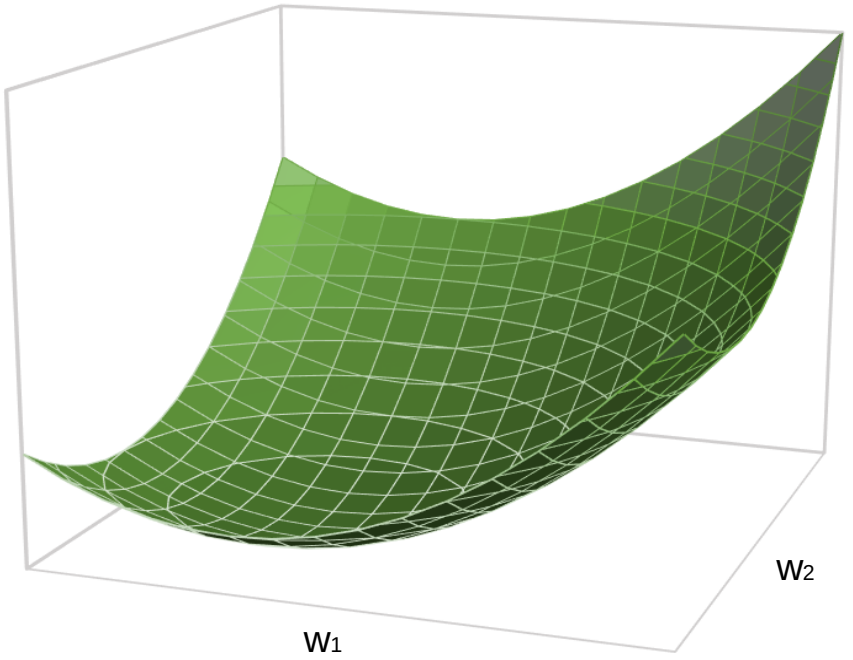
$$l = \sum_i | y_i - o_i | = || y - o ||_1$$

  - Summe der Fehlerquadrate (L2 Norm)

$$l = \sum_i (y_i - o_i)^2 = || y - o ||_2^2$$

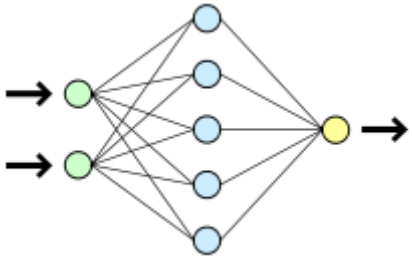
- Gradientenabstieg (gradient descent)

x_1	x_2	y_1	o_1	l
0	0	0	0	
0	1	0	0	
1	0	0	1	
1	1	1	0	

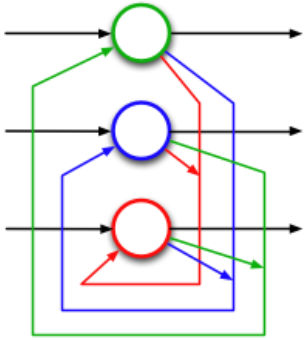


Netzwerkstrukturen

- Feed forward
  - Verbindungen nur zur nächsten Schicht
  - Keine Zykel, kein Überspringen von Schichten
  - Berechnet eine Funktion seiner Inputs, kein interner Zustand
  - Einschichtig oder Mehrschichtig
  - Beispiele: Perceptron, Backpropagation Netzwerke



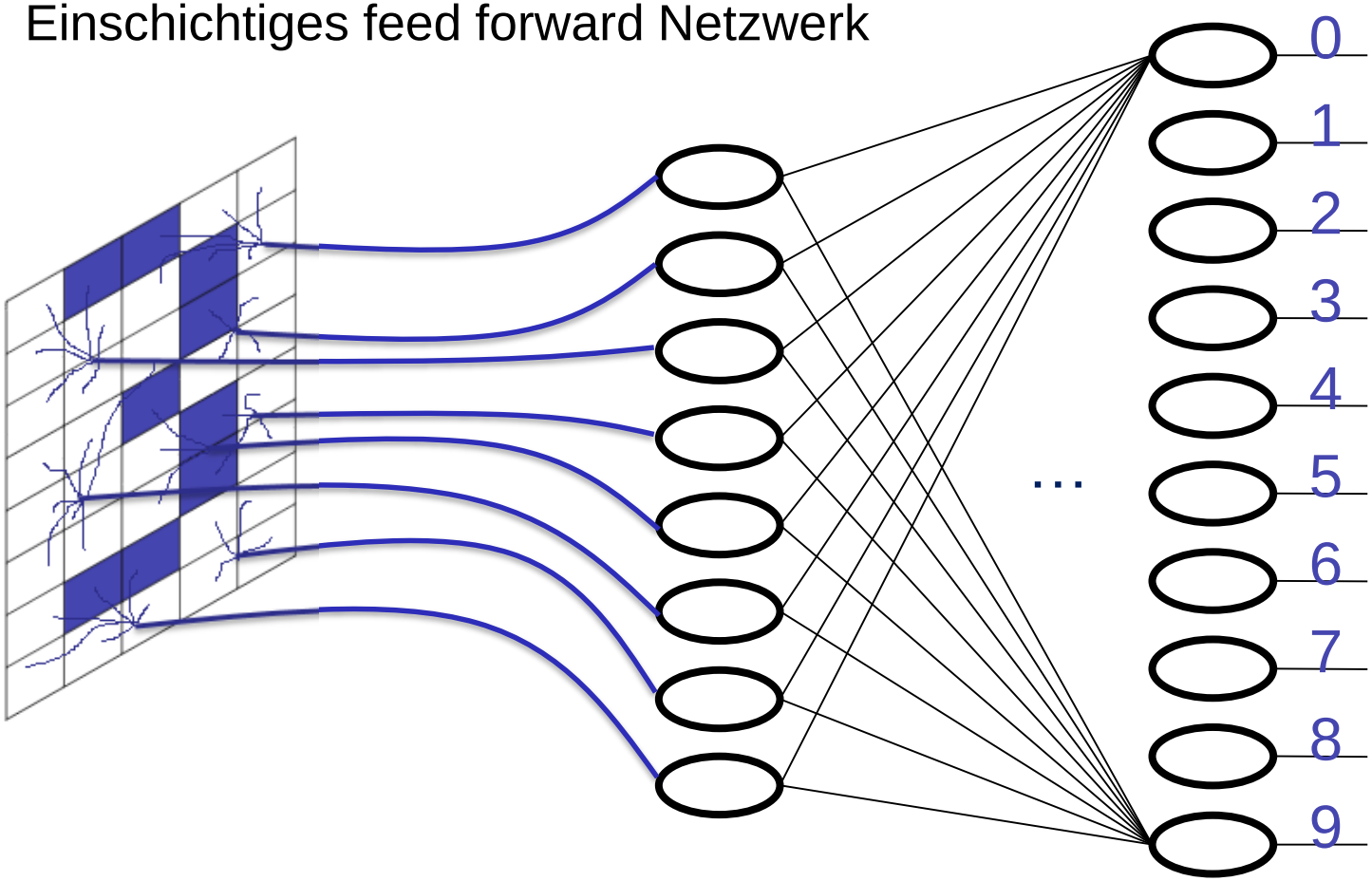
- Recurrent
  - Jede Topologie erlaubt, inklusive Rückkopplung
  - Zustand steckt nicht nur in den Gewichten, sondern auch in der Aktivierung
  - Beispiele: Hopfield Netzwerke, Boltzmann Maschinen, Echo State Netzwerk, LSTM



- Selbstorganisierende Netzwerke
  - Beispiel: Kohonen Netzwerk

Perceptron

- Von Frank Rosenblatt 1958 vorgestellt
- Einschichtiges feed forward Netzwerk



Perceptron

- Berechnung der Outputs

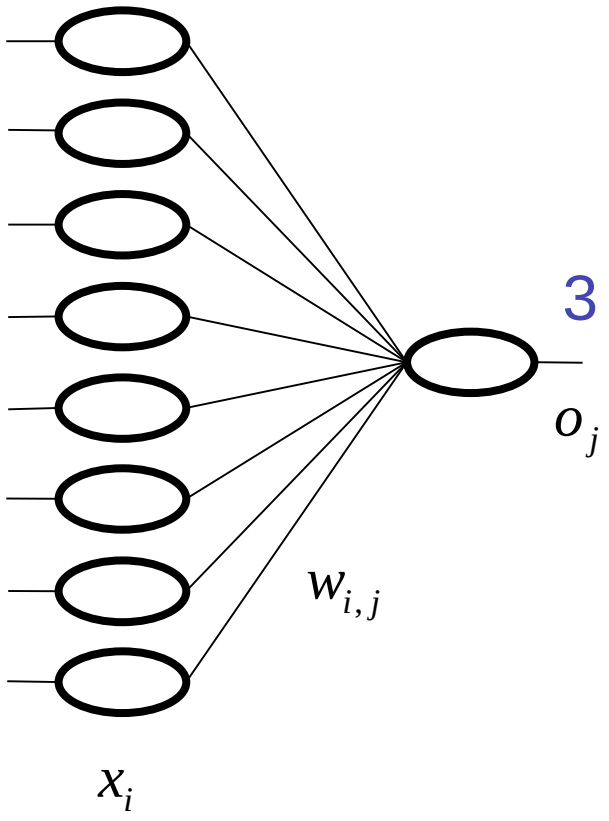
$$o_j = Step_0(\sum_{i+1} w_{i,j} x_i)$$

- Lernregel (Delta Regel)

$$w_{i,j} = w_{i,j} + \alpha \cdot x_i \cdot (y_j - o_j)$$

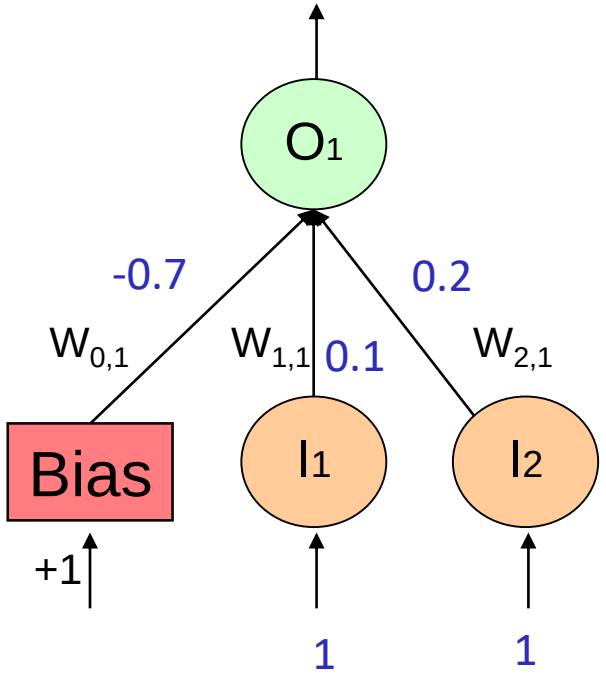
- Algorithmus

```
initialisiere Gewichte zufällig
do
    for each e in examples
     berechne Output
     passe Gewichte an
while (Fehler zu groß und Abbruch-
       kriterium nicht erreicht)
```

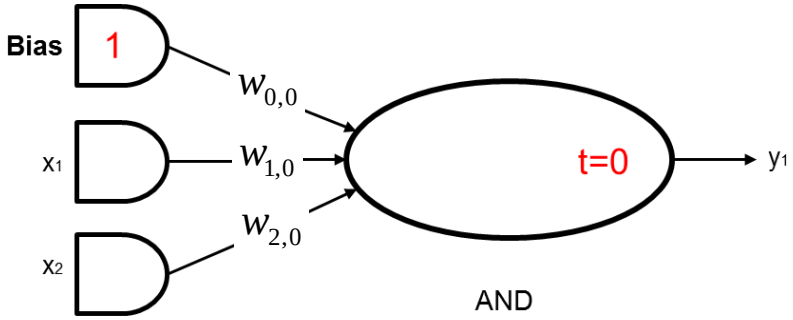


Perceptron

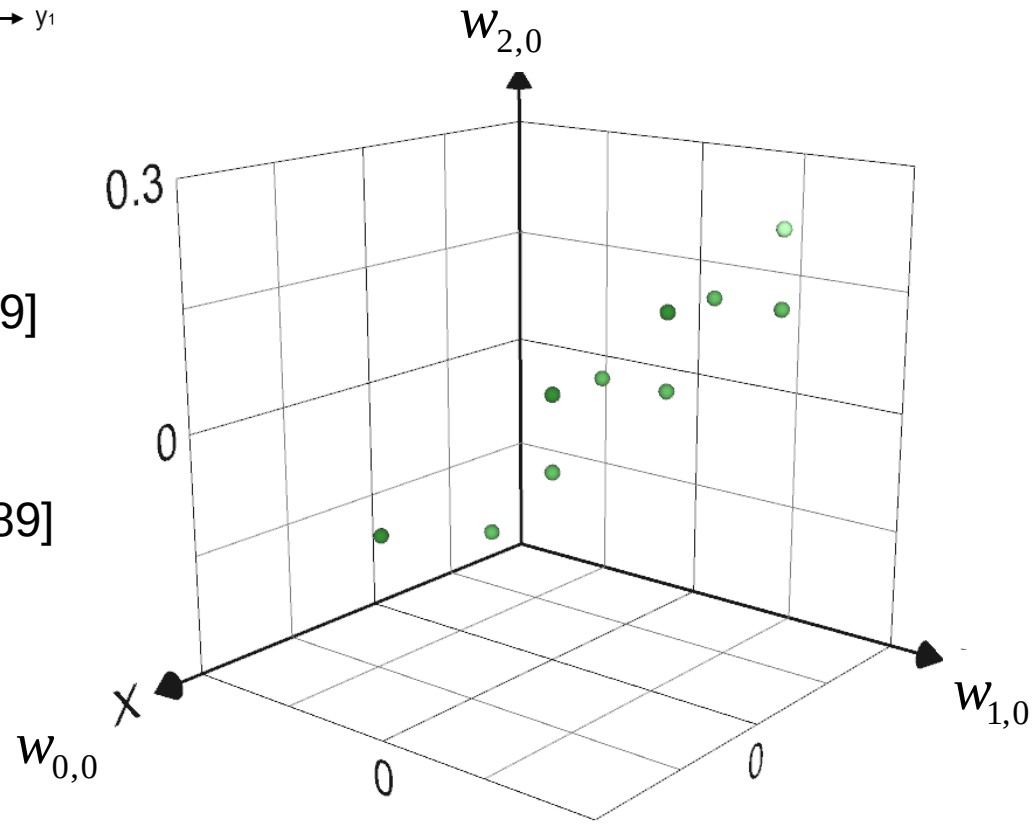
- Beispiel
  - Input: (1,1)
  - Gewichte: $W_{0,1}$=-0.7; $W_{1,1}$=0.1; $W_{2,1}$=0.2;
  - Output:
    - o_1=Step$_0$(1*(-0.7) + 1*0.1 + 1*0.2)
    - o_1=Step$_0$(-0.4) = 0
  - Zieloutput: 1
  - Lernfaktor α: 0.9
  - Gewichtsanpassung:
    - $W_{0,1}$=-0.7 + 0.9 * 1 * (1 – 0) = 0.2
    - $W_{1,1}$=0.1 + 0.9 * 1 * (1 – 0) = 1.0
    - $W_{2,1}$=
  - Output jetzt:
    - o_1=



Perceptron
And Funktion



- Zufälliger Anfang
  - $w_{x,0}$ [0.028; -0.258; -0.189]
  - 3 Muster falsch
- 4 Muster anlegen und lernen
  - $w_{x,0}$ [-0.072; -0.158; -0.189]
  - 2 Muster falsch
- …
  - 0 Muster falsch



Perceptron
Lernrate und Gradientenabstieg

- Kleine Lernrate

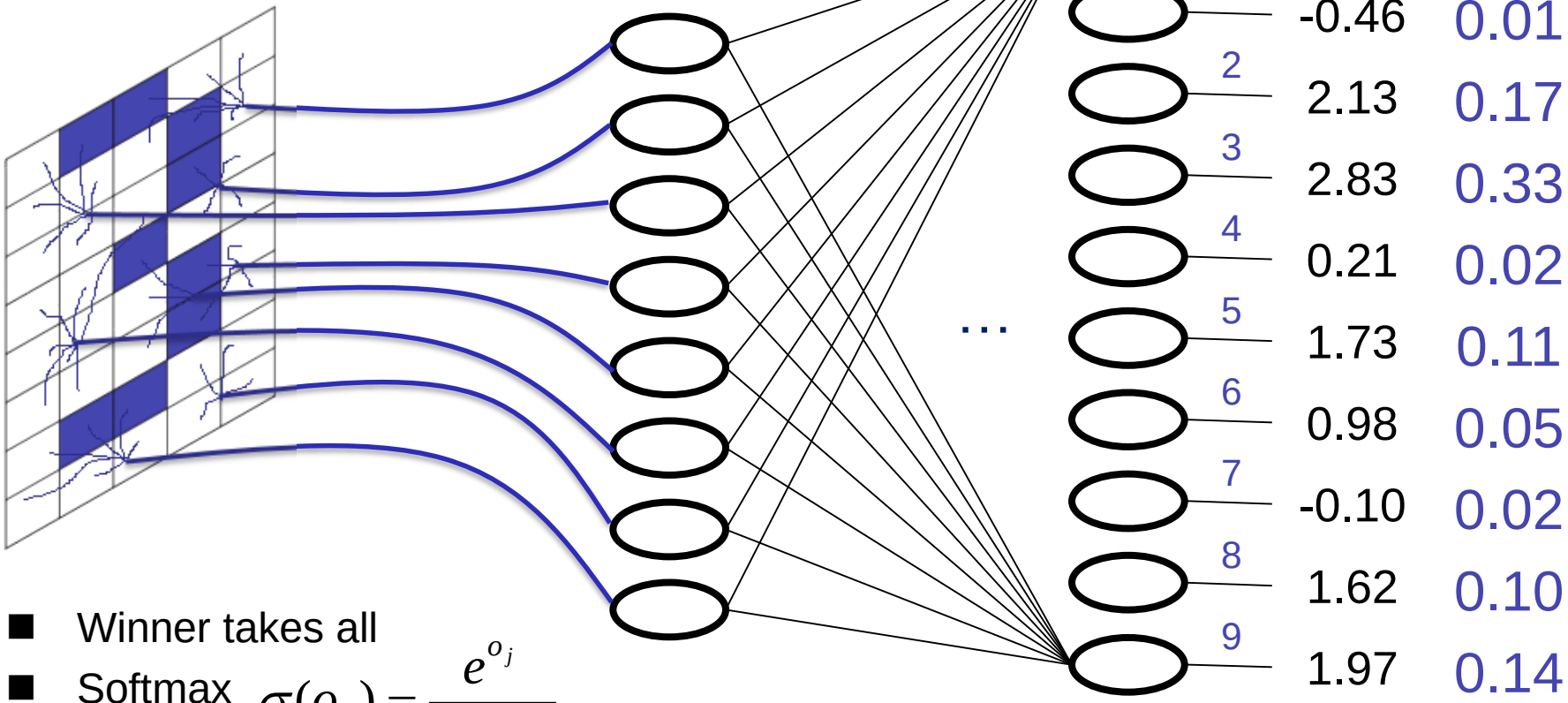
- Große Lernrate

- Stochastic Gradient Descent
- Batch Learning

Perceptron

Reelwertige Inputs und Outputs

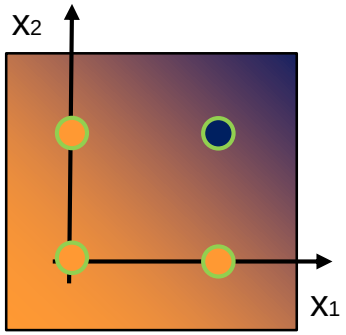
$$o_j = Step_0(\sum_{i+1} w_{i,j} x_i)$$



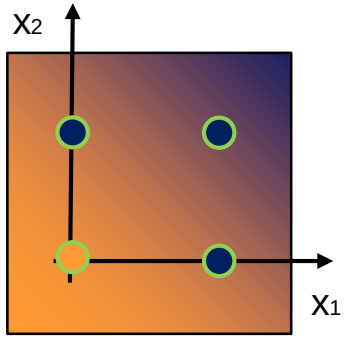
- Winner takes all
- Softmax $\sigma(o_j) = \frac{e^{o_j}}{\sum_k e^{o_k}}$

Perceptron
Ausdrucksfähigkeit

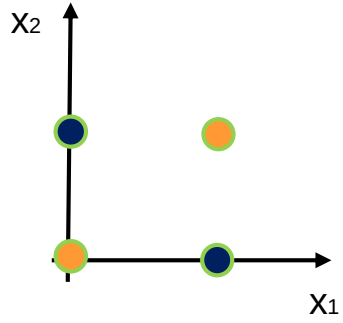
- Kann nur linear separierbare Probleme repräsentieren



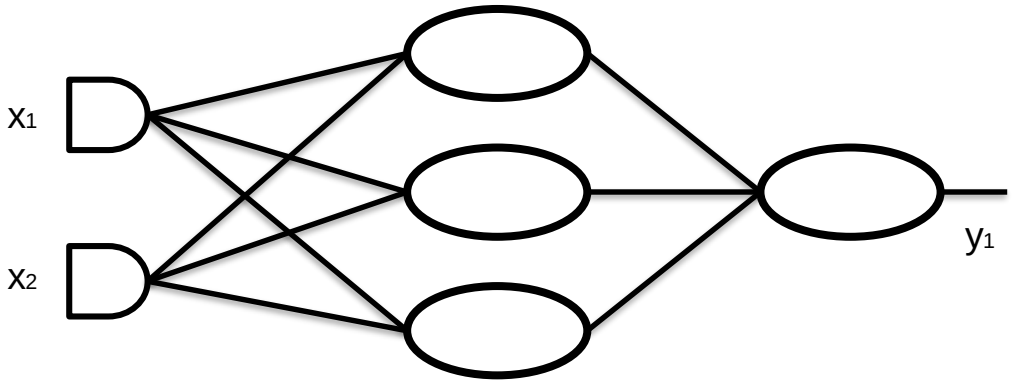
AND



OR



XOR



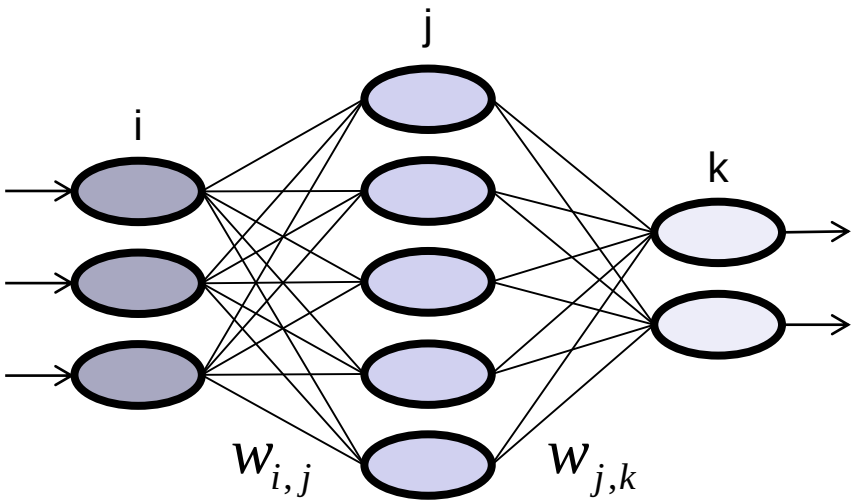
Backpropagation Networks

- Um XOR oder ähnliche Probleme zu lernen benötigt man
  - Mehrschichtige Netzwerke
  - Eine nicht-lineare Aktivierungsfunktion
- Problem
  - Wie propagiere ich den Fehler an den Outputs zurück in die Hidden Units und passe deren Gewichte an?
- Lösung
  - Backpropagation of Error
  - Bryson & Ho 1969, Rumelhart, Hinton & Wiliams 1986

Backpropagation Networks
Lernregel

- Berechnung der Outputs

$$o_j = \sigma(\sum_{i+1} w_{i,j} x_i)$$



- Lernregel
  - Gewichte zur Outputschicht

$$w_{j,k} = w_{j,k} + \alpha \cdot o_j \cdot \Delta_k \text{ mit } \Delta_k = \sigma'(in_k) \cdot (y_k - o_k)$$

  - Gewichte zu Hiddenschichten

$$w_{i,j} = w_{i,j} + \alpha \cdot o_i \cdot \Delta_j \text{ mit } \Delta_j = \sigma'(in_j) \cdot \sum_k w_{j,k} \Delta_k$$

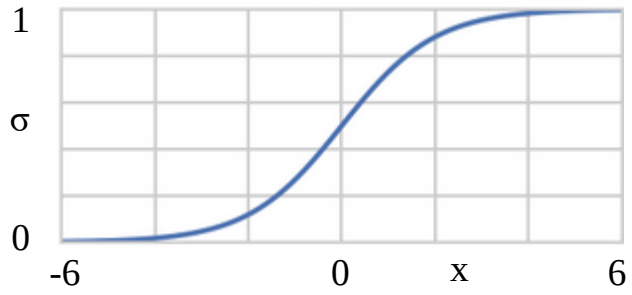
Backpropagation Networks
Algorithmus

- Algorithmus

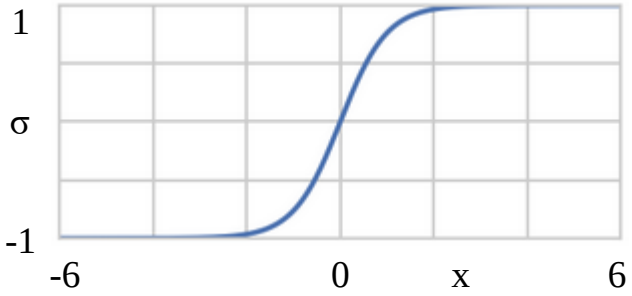
```
initialisiere Gewichte zufällig
do
   for each e in examples
    berechne Output (recall)
    berechne Δ Werte für die Output Units
    wiederhole für jede Schicht (rückwärts von den
       outputs)
       propagiere Δ Werte zurück zur vorigen Schicht
       berechne neue Gewichte
while (Fehler zu groß und Abbruchkriterium nicht erreicht)
```

Backpropagation Networks
Aktivierungsfunktion

- Sigmoid
  - Funktion $\sigma(x) = \frac{1}{1+e^{-x}}$
  - Ableitung $\sigma'(x) = \sigma(x)(1-\sigma(x))$

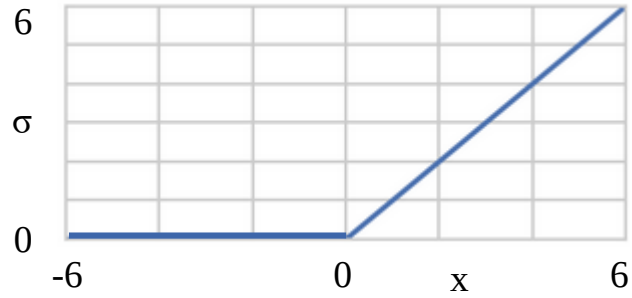


- Tanh $\sigma(x) = \tanh(x)$



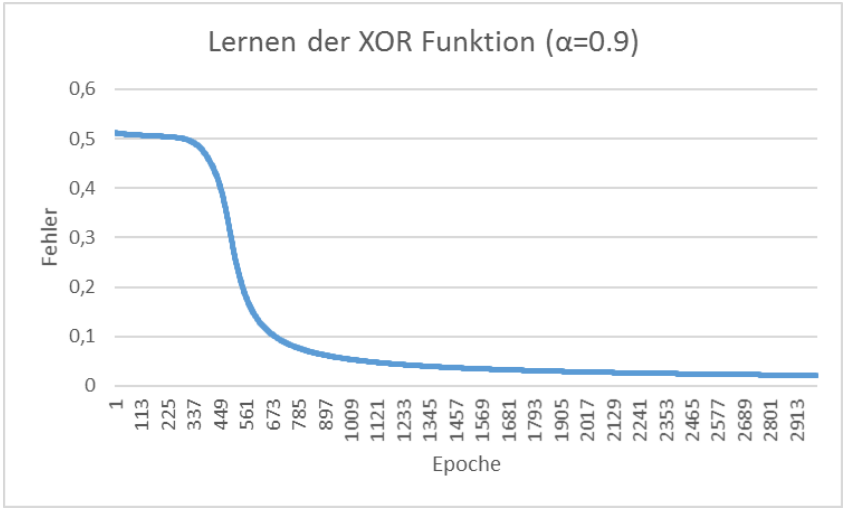
- Relu $\sigma(x) = \max(0, x)$

  (Rectified linear unit)

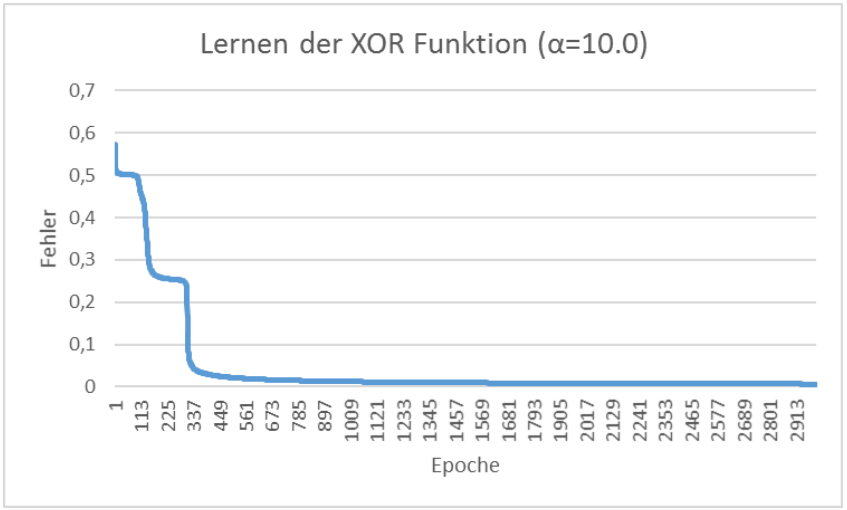


Backpropagation Networks
Beispiel: XOR Funktion

- Ein 3 – 3 – 1 Netzwerk kann das XOR Muster lernen
  - Lernrate 0.9

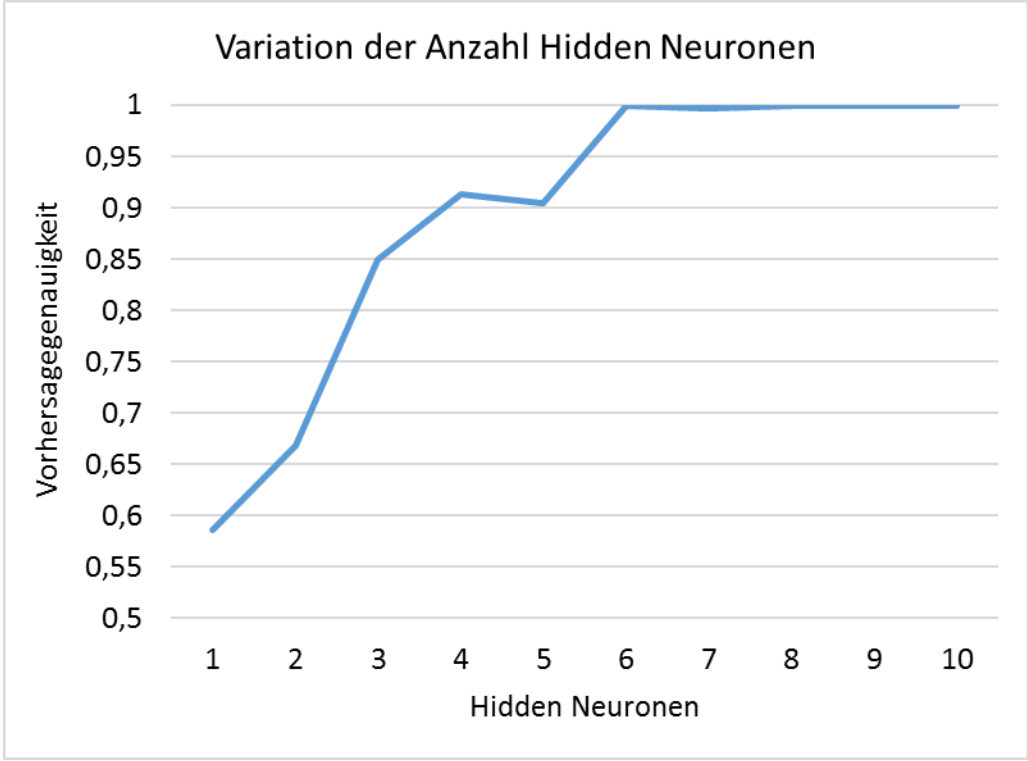


  - Lernrate 10



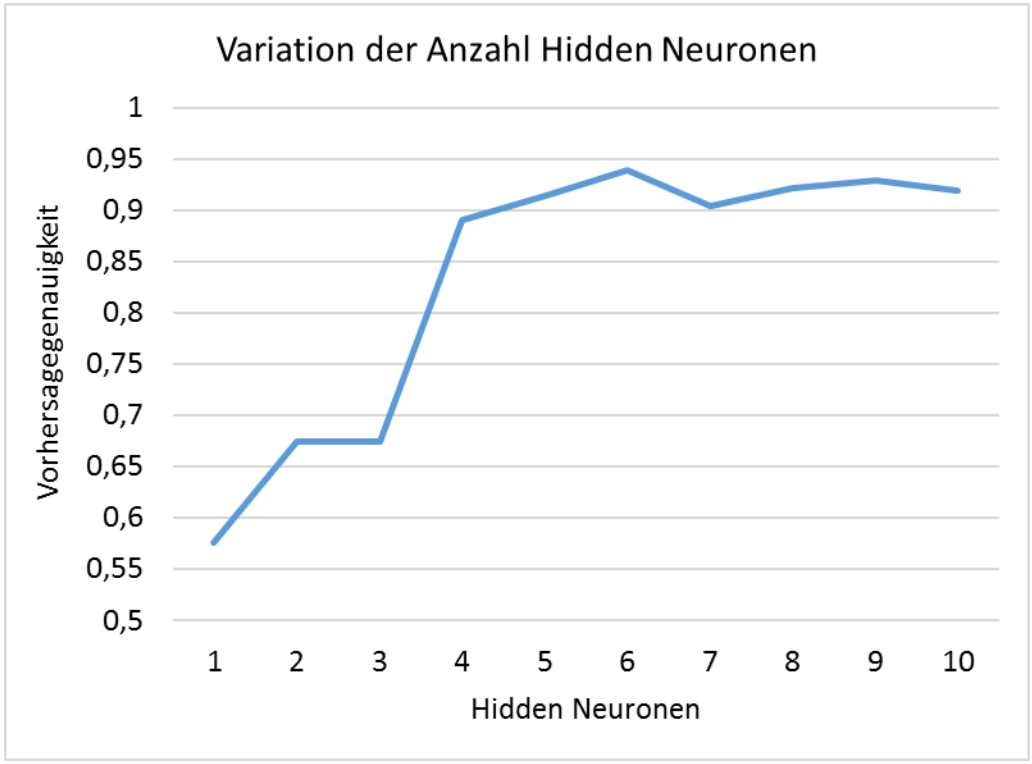
Backpropagation Networks
Beispiel: Klassifikation

- Wie groß muss das Netzwerk sein?
  - Um die Trainingsmuster zu lernen?
  - Test der Vorhersagegenauigkeit auf den Trainingsdaten
- Datensatz
  - 864 Beispiele Training
  - 864 Beispiele Test
  - 6 Attribute (4,4,4,3,3,3)
  - Ziel: ja/nein
- Netzwerk
  - 21 Input Neuronen
  - x Hidden Neuronen
  - 2 Output Neuronen



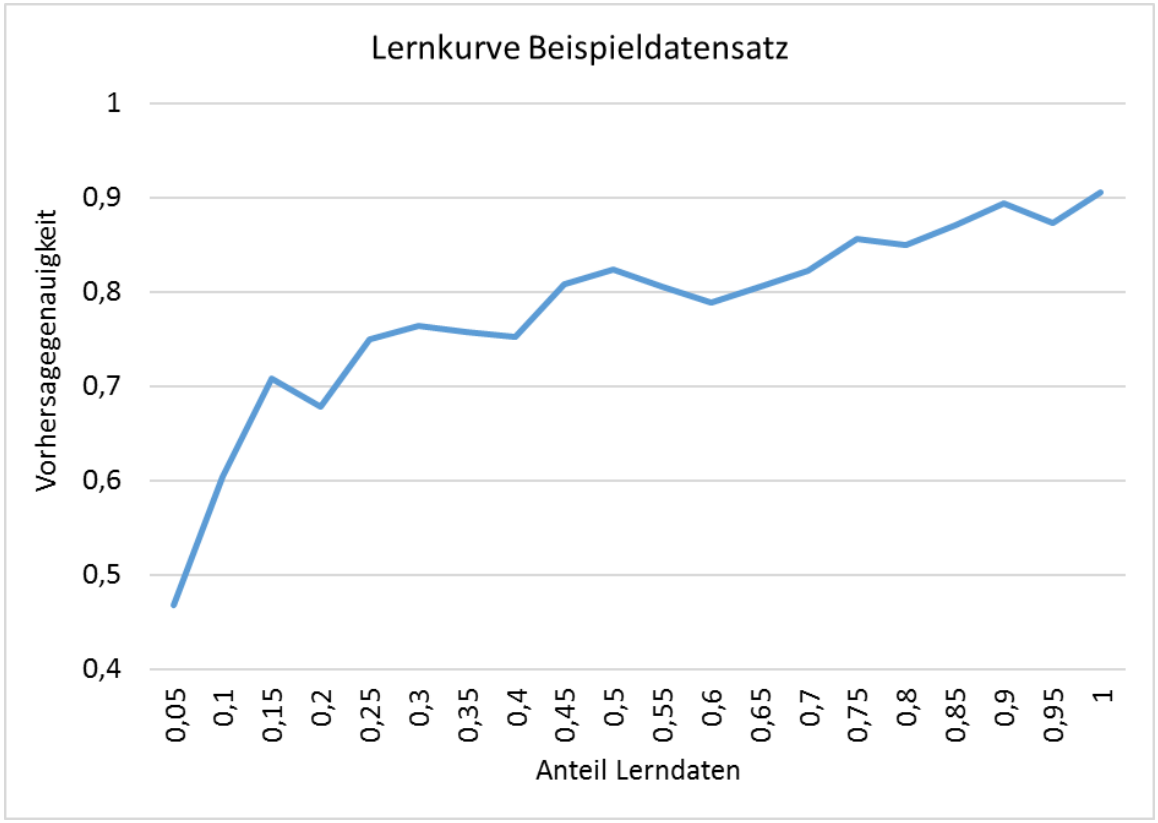
Backpropagation Networks
Generalisierung

- Wie groß muss das Netzwerk sein?
  - Um unbekannte Muster richtig zu klassifizieren?
  - Test der Vorhersagegenauigkeit auf den Testdaten

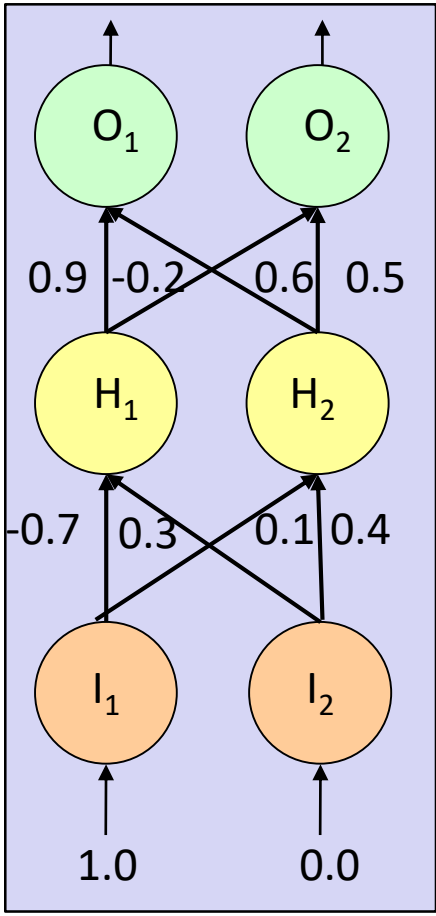


Backpropagation Networks
Lernkurve

- Wie viele Beispiele benötigt das Netzwerk, um zu generalisieren?
  - Je mehr desto besser



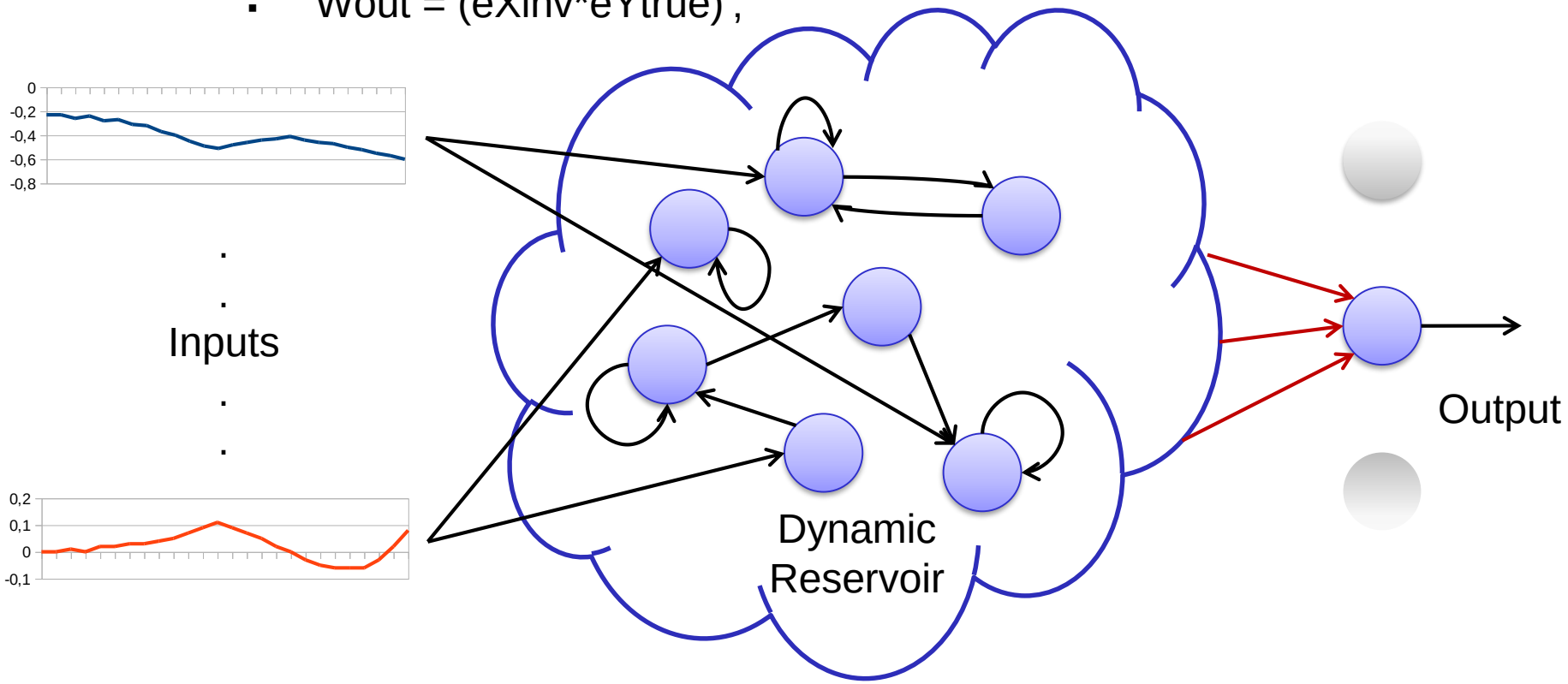
Backpropagation



- Input: {1, 0}, Zieloutput: {1, 0}, g() sigmoidal, $\alpha = 0.9$
- Recall
  - $a_{H1} = g(-0.7*1.0 + 0.3 * 0.0) = 0.332$
  - $a_{H2} = g(0.1*1.0 + 0.4 * 0.0) = 0.525$
  - $o_1 = g(0.9 * 0.332 + (-0.2) * 0.525) = 0.548$
  - $o_2 = g(0.6 * 0.332 + 0.5 * 0.525) = 0.613$
- Training:
  - $\Delta_{O1}=(0.548 * (1 - 0.548)) *(1–0.548) = 0.248 * 0.452 = 0.112$
  - $W_{H1,O1}= 0.9 + 0.9 * 0.332 * 0.112 = 0.9 + 0.033 = 0.933$
  - $W_{H2,O1}= -0.2 + 0.9*0.525 * 0.112 = -0.2 + 0.053 = -0.147$
  - $\Delta_{O2}=(0.613 * (1 - 0.613)) *(0–0.613) = 0.237 *(-0.613)= -0.145$
  - $W_{H1,O2}= 0.557$ $W_{H2,O2}= 0.431$
  - $\Delta_{H1} = (0.332 * (1 – 0.332)) * (0.9 * 0.112 + 0.6 * (-0.145)) =0.222 * 0.014 = 0.003$
  - $W_{1,1}= -0.7 + 0.9 * 1.0 * 0.003 = -0.697$
  - $W_{2,1}= 0.3 + 0.9 * 0.0 * 0.003 = 0.3$
  - $\Delta_{H2} = (0.525 * (1-0.525)) * (-0.2 * 0.112 + 0.5 * (-0.145)) =0.249 * (-0.095) = -0.024$
  - $W_{1,2}= 0.079$ $W_{2,2}= 0.4$
- Output jetzt: $o_1= 0.558$ $o_2= 0.600$

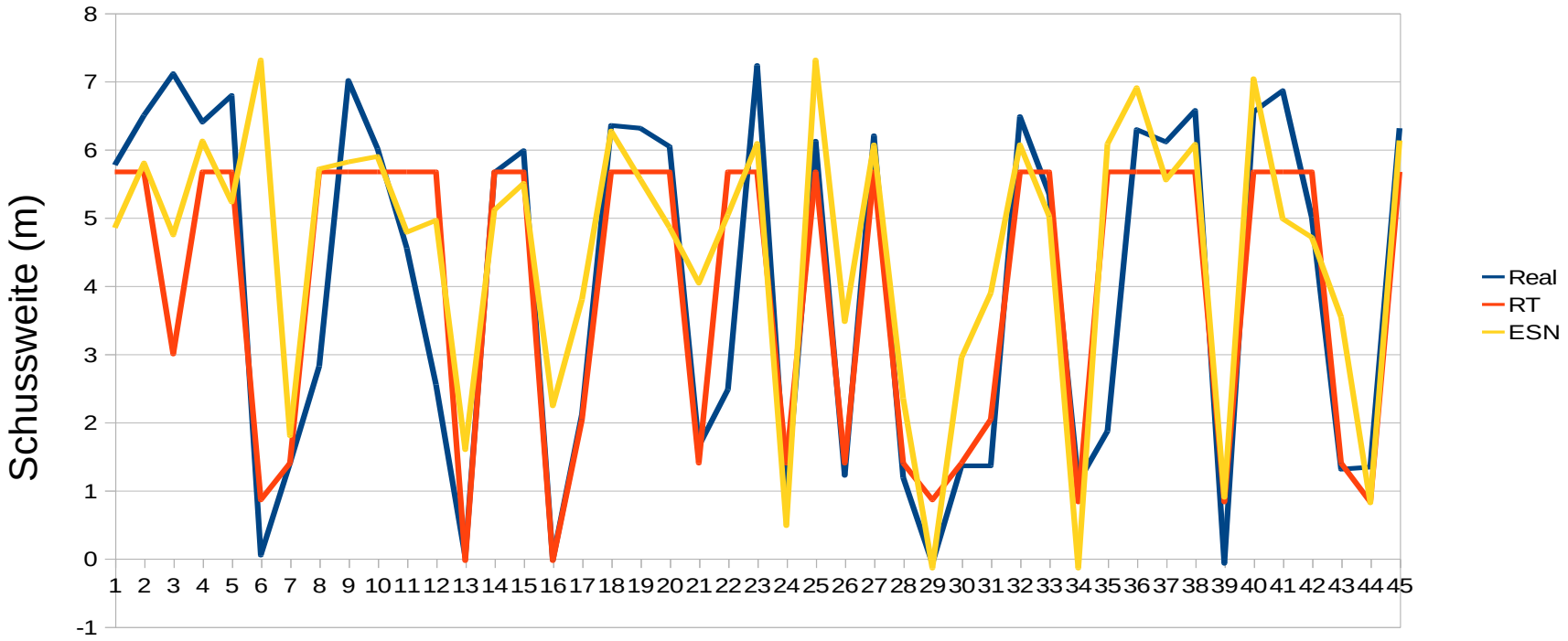
Echo State Netzwerke

- Rekurrente neuronale Netzwerke
  - Reservoir Neuronen sind zufällig sparsly connected
  - Gelernt werden nur die Gewichte zum Output
    - eXinv = pinv(eX);
    - Wout = (eXinv*eYtrue)';



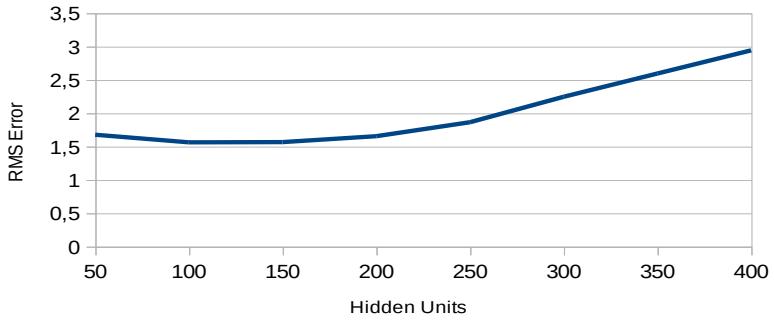
Echo State Netzwerke

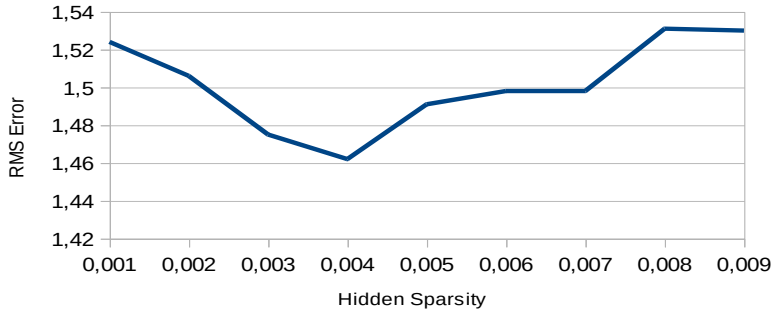
- Beispiel
  - Input: Gyro, Ballposition, Gyroänderung als Zeitserien
  - Output: Schussweite
- Vergleich
  - Regression Tree, Echo State Netzwerk, Real

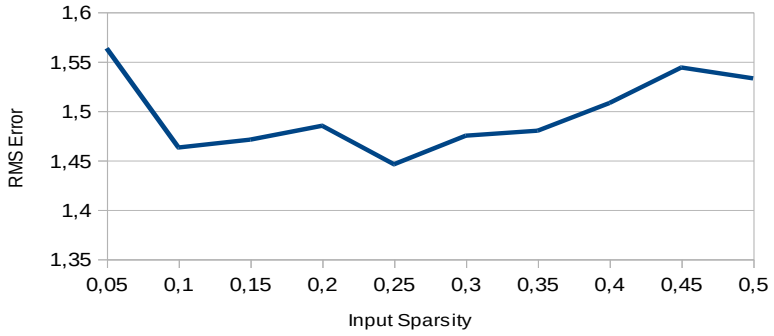


Echo State Netzwerke

- Ergebnisqualität (in diesem Beispiel)
  - RMSE 1,67
  - RT 1,50
  - Aber Kombination aus beiden liefert RMSE 1,31
- Vorteile
  - Berücksichtigen zeitlichen Verlauf des Inputs
- Nachteile
  - Mehr freie Parameter
  - Höherer Rechenaufwand

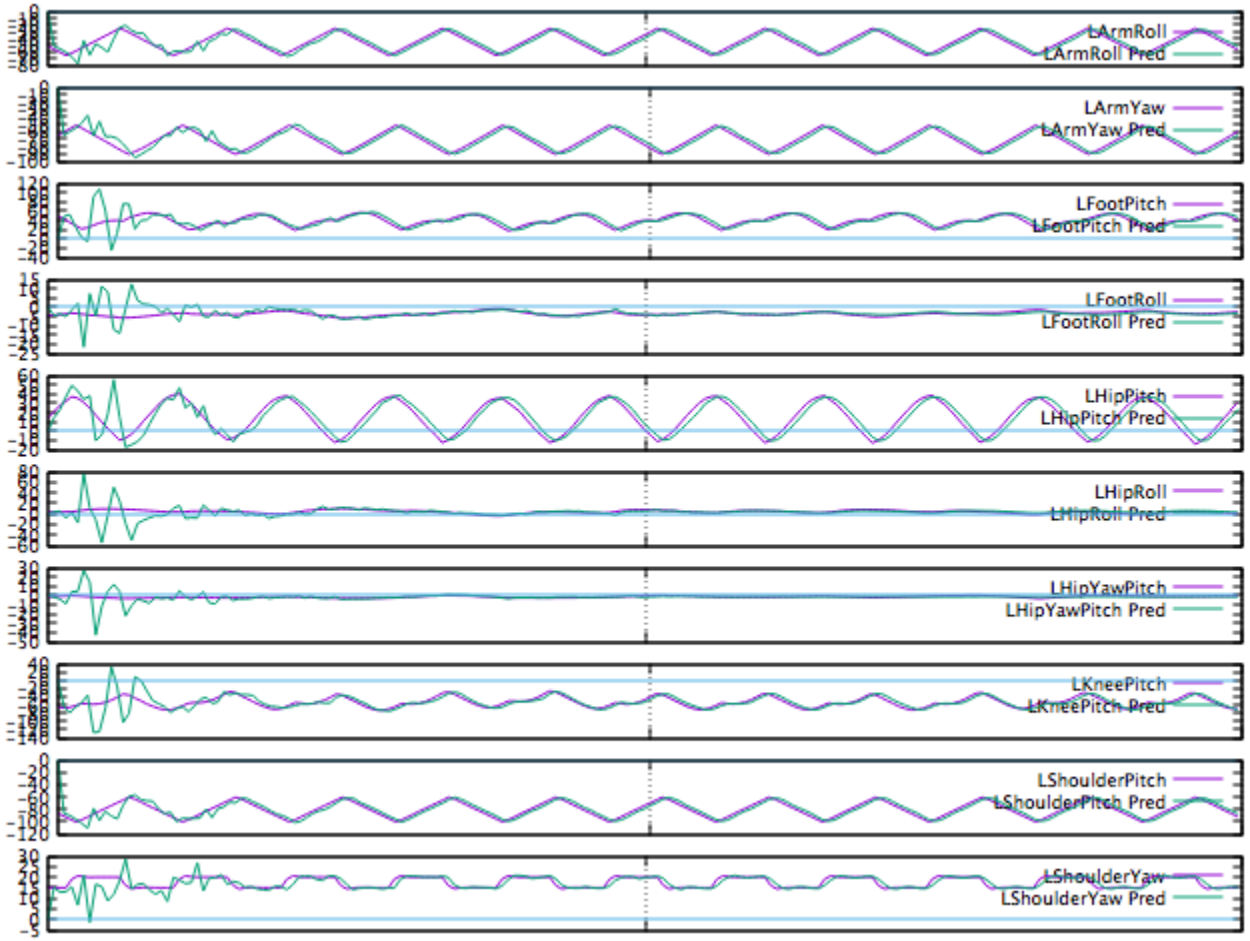






Echo State Netzwerke

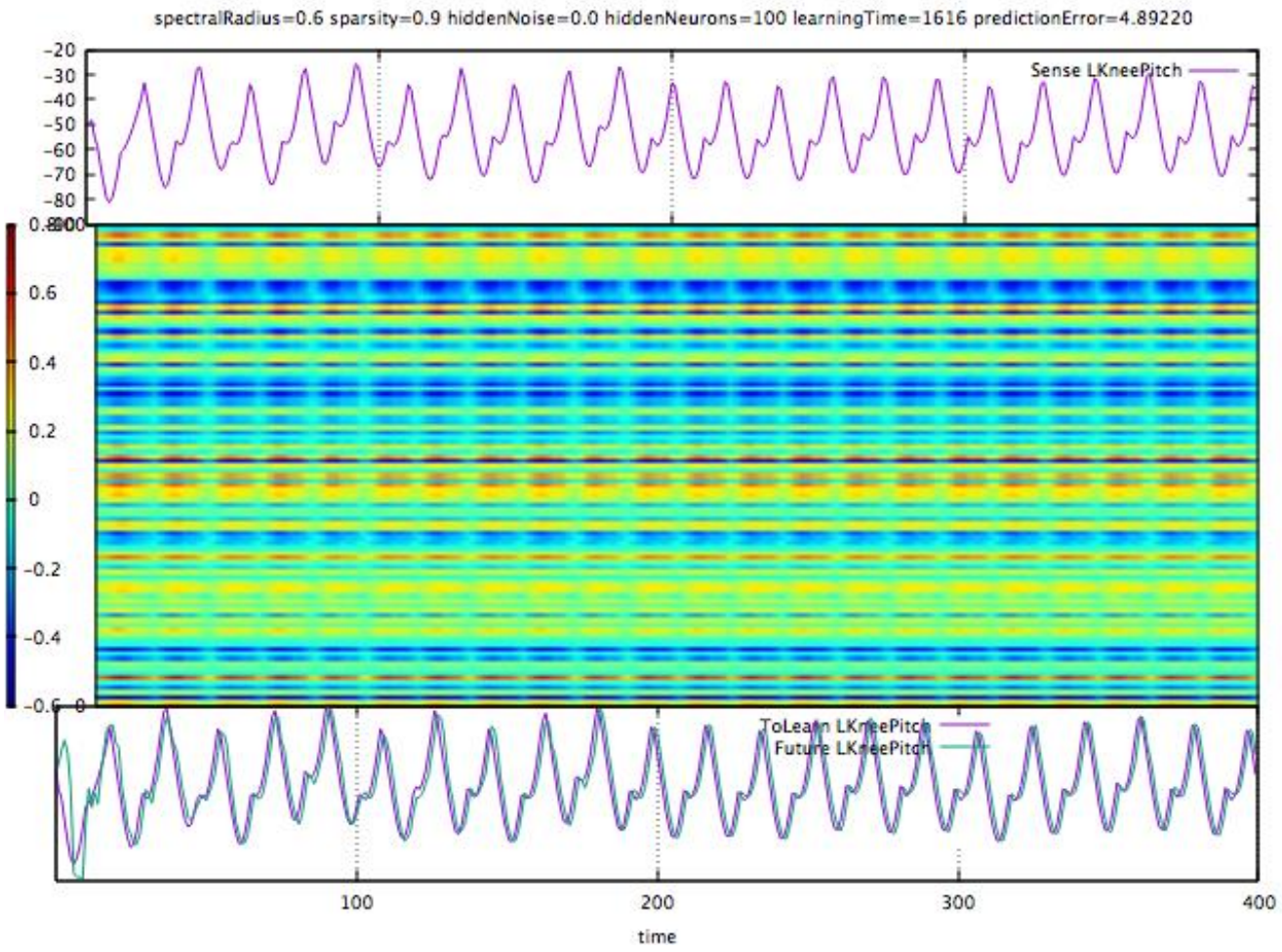
- Beispiel
  - Lernen der Gelenkkurven beim Laufen auf zwei Beinen
  - Input: Sinuskurve
  - Output: Gelenkkurven



Quelle: Viktor Kurz

Echo State Netzwerke

- Aktivierung der hidden Neuronen über die Zeit



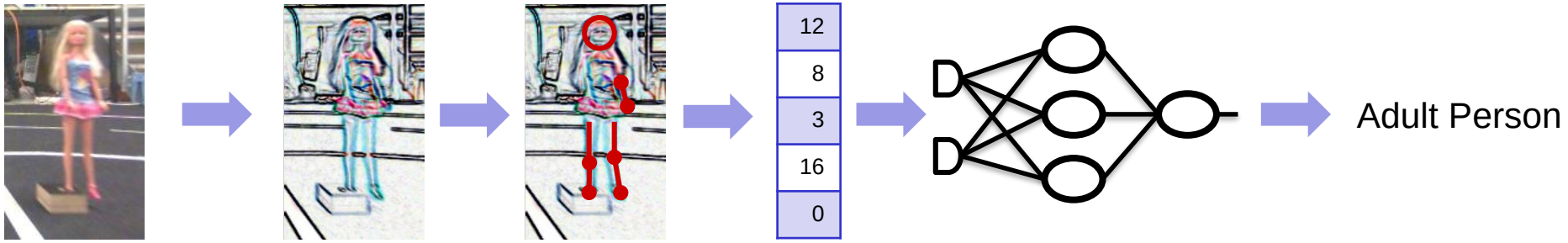
Quelle: Viktor Kurz

Warum ist Deep Learning so aktuell?

- Mehr Daten
  - Youtube: jede Minute werden 500 Stunden Video hochgeladen
  - Facebook: täglich werden 300 Millionen Bilder hochgeladen
- Mehr Rechenpower
  - GPU Beschleunigung
  - GPU/CPU Cluster
- 'Neue' Deep Learning Ansätze
  - Deep Neural Networks
  - Convolutional Neural Networks
  - Bessere Aktivierungsfunktionen, Optimizer, Initialisierung, …
- Frei verfügbare Frameworks
  - Theano, TensorFlow, DeepLearning4J, …

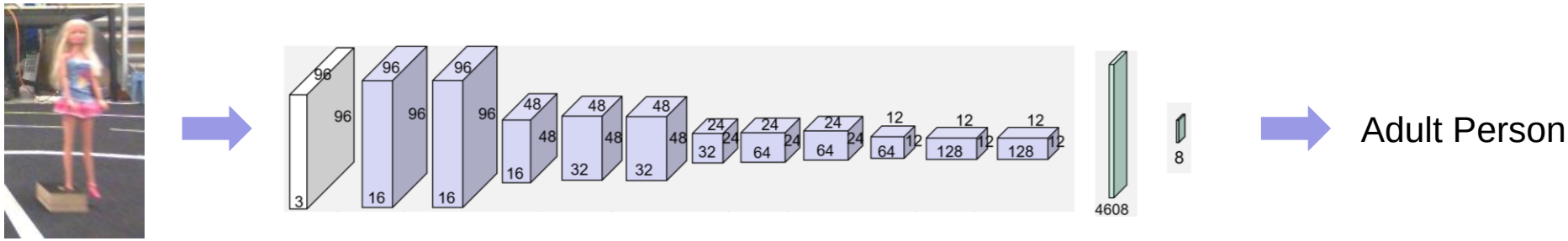
Deep Learning

- Ohne

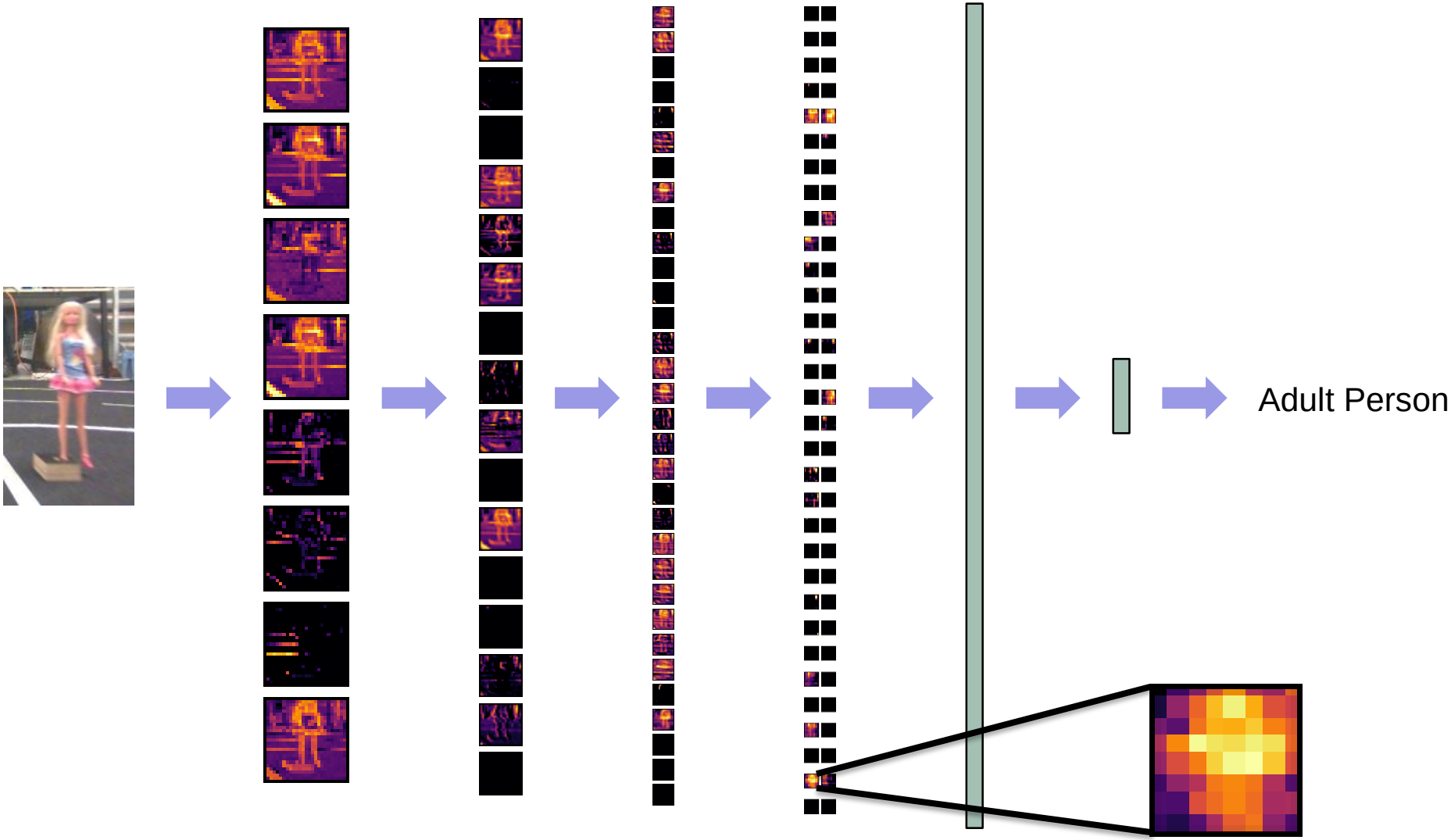


Feature Engineering

- Mit



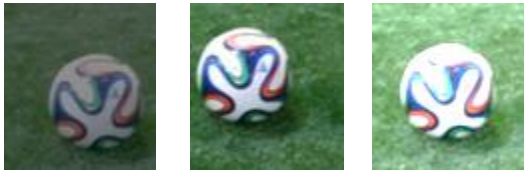
Deep Learning



Convolutional Neural Networks

- Sweaty will Fußball spielen
- Dazu muss er Objekte auf Bildern erkennen
- Es ist egal, wo im Bild ein Gegenstand ist

- Es soll möglichst egal sein, wie hell es ist

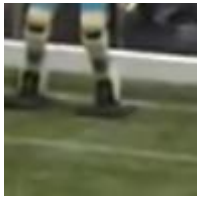


- Es soll egal sein, wie der Ball aussieht

Convolutional Neural Networks

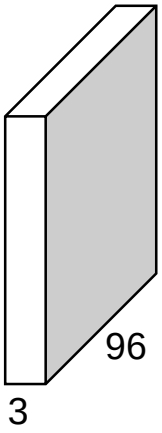
- Input ist 3D
  - Unser Beispiel: 96x96 Pixel, 3 Kanäle (Red, Green, Blue)





- Output ist Feature-Vektor
  - Ball X-Linie Roboter L-Linie Obstacle

Input



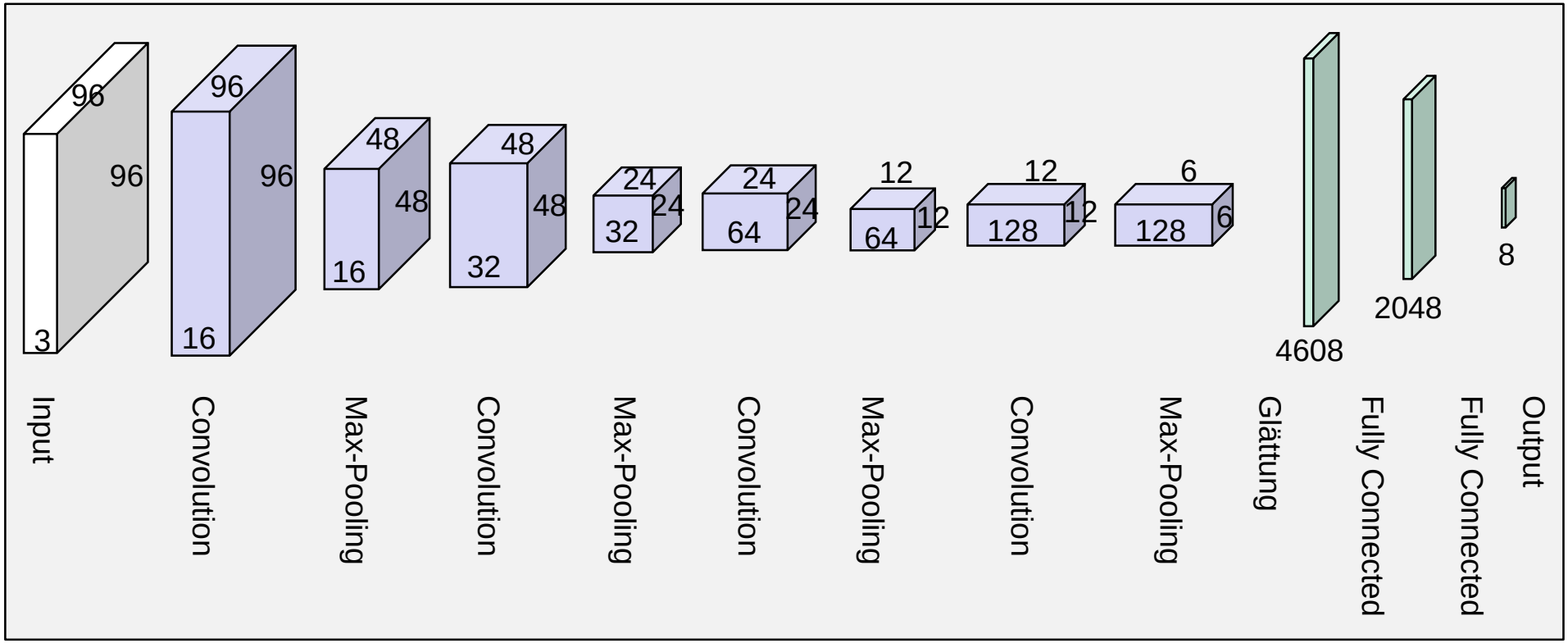
Netzwerk

Ball
Pfosten
Obstacle
L
X
T
Punkt
Fuß

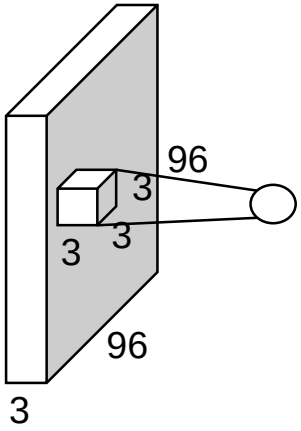
8

Convolutional Neural Networks

Beispielarchitektur



Convolutional Neural Networks
Convolution Layer



- Filter wandert über Input
- Berechnet Feature Input für Neuron in Activation Map
- Aktivierungsfunktion: ReLU
  - Y = ReLU(Wx+b)

1	0	1
0	1	0
1	0	1

W

1	1	1	0
0	1	1	1
0	0	1	1
0	0	1	1

x

0	0	0	0
0	0	0	0
0	0	0	0
1	0	0	1

b

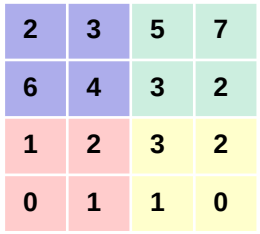
	4		

Y

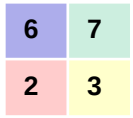
- Wird mit n Filtern wiederholt: n Activation Maps
- Padding: Wie wird mit dem Rand umgegangen
- Strides: Schrittweite, > 1 bedeutet Activation Map wird kleiner

Convolutional Neural Networks
Pooling Layer

- Reduziert die Größe der Activation Map
- Poolingfunktion
  - Meist wird Max-Pooling verwendet
  - Alternative Avg-Pooling
- Reduzieren die Informationsmenge
- Beispiel
  - 2x2 Filter mit 2,2 strides



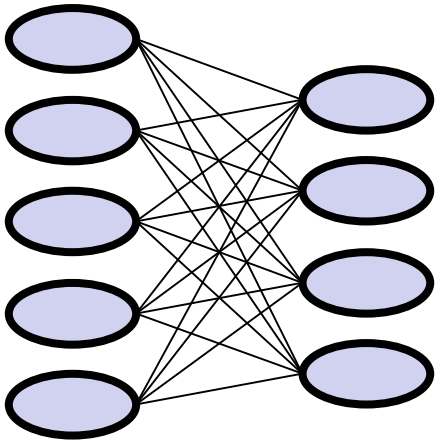
Vorher



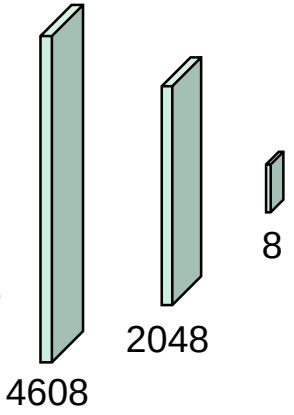
Nachher
Max- Pooling

Convolutional Neural Networks
Fully Connected Layer

- Bereits bekannt: Backpropagation Schichten
- Manchmal genügt es, für ein neues Bilderkennungs-problem nur diese Schichten neu zu lernen
- Hier stecken die meisten lernbaren Gewichte



- Beispiel Ballerkennung
  - 4608 * 2048 = 9.437.184 Gewichte
  - 2048 * 8 = 16.384 Gewichte
- Zum Vergleich
  - In der ersten Convolution Schicht sind 27*16 Gewichte



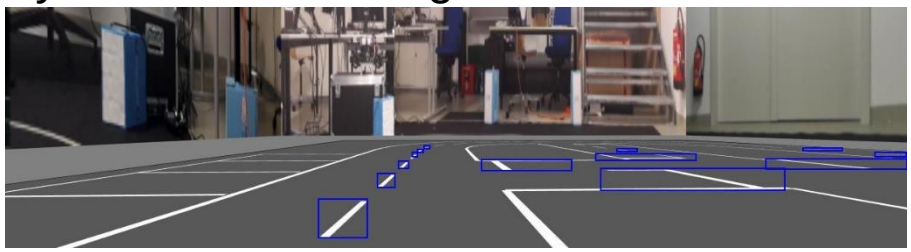
Convolutional Neural Networks
Lernen

- Bilder taggen
  - Für jedes Bild wird die richtige Klassifikation gespeichert
  - Gegebenenfalls eine bounding box
- Bilder lernen
  - Trainingsdatensatz ans Netzwerk anlegen
  - Netzwerk berechnet Output und Fehler
  - Netzwerk ermittelt mit Gradientenabstieg Gewichtsanpassungen, um den Fehler zu reduzieren
- Qualität auf Testdatensatz überprüfen
- Anwenden
  - Gespeichertes Netzwerk auf Live Bildern betreiben
  - Schneller Recall

Convolutional Neural Networks
Overfitting

- Viele Bilder zeigen
- Wenn nicht genügend vorhanden?
  - Vorhandene Bilder mehrfach verwenden

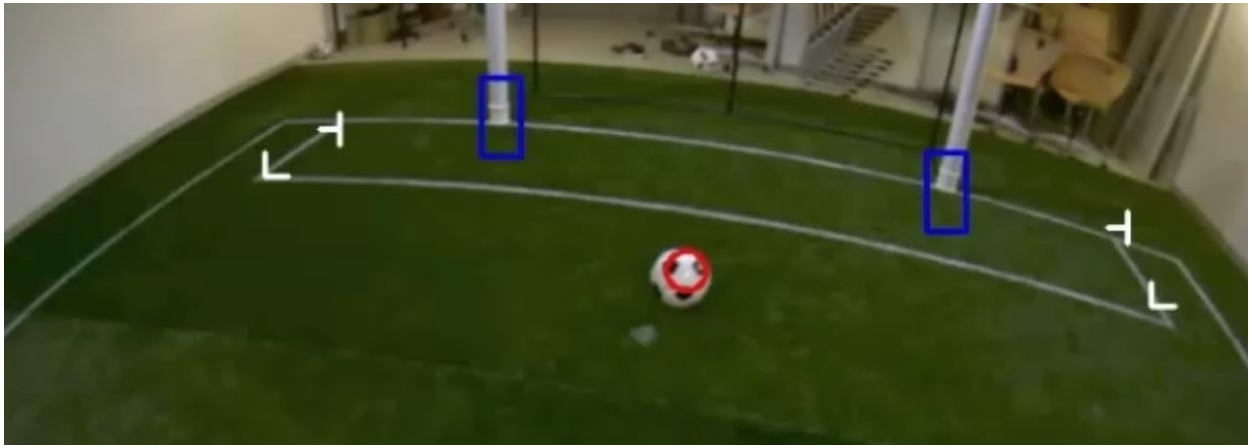
  - Synthetische Bilder generieren



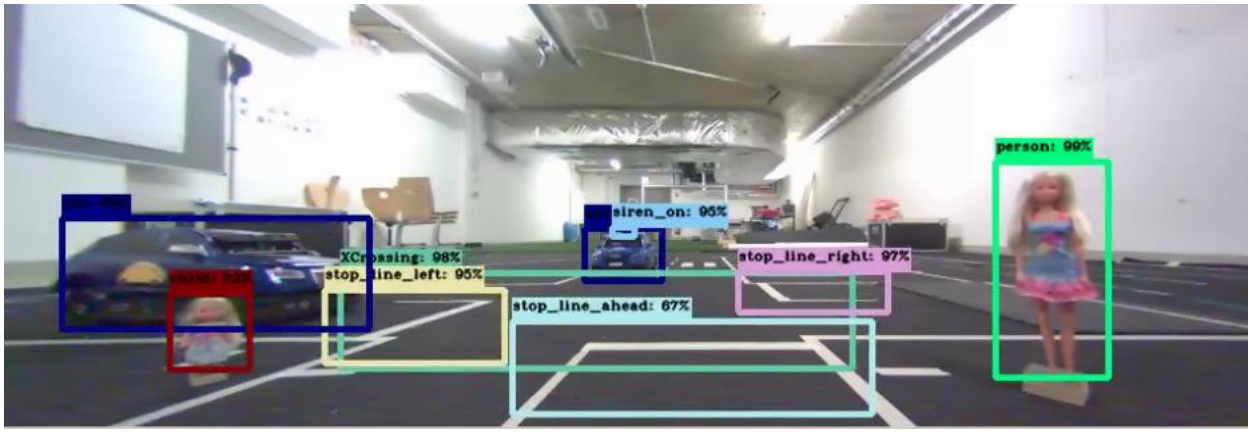
- Dropout
  - Beim Lernen einen bestimmten Prozentsatz zufällig ausgewählter Verbindungen nicht berücksichtigen
  - Vermeidet einzelne ‚wichtige' Verbindungen

Deep Learning
Beispiel Hochschule

- RoboCup

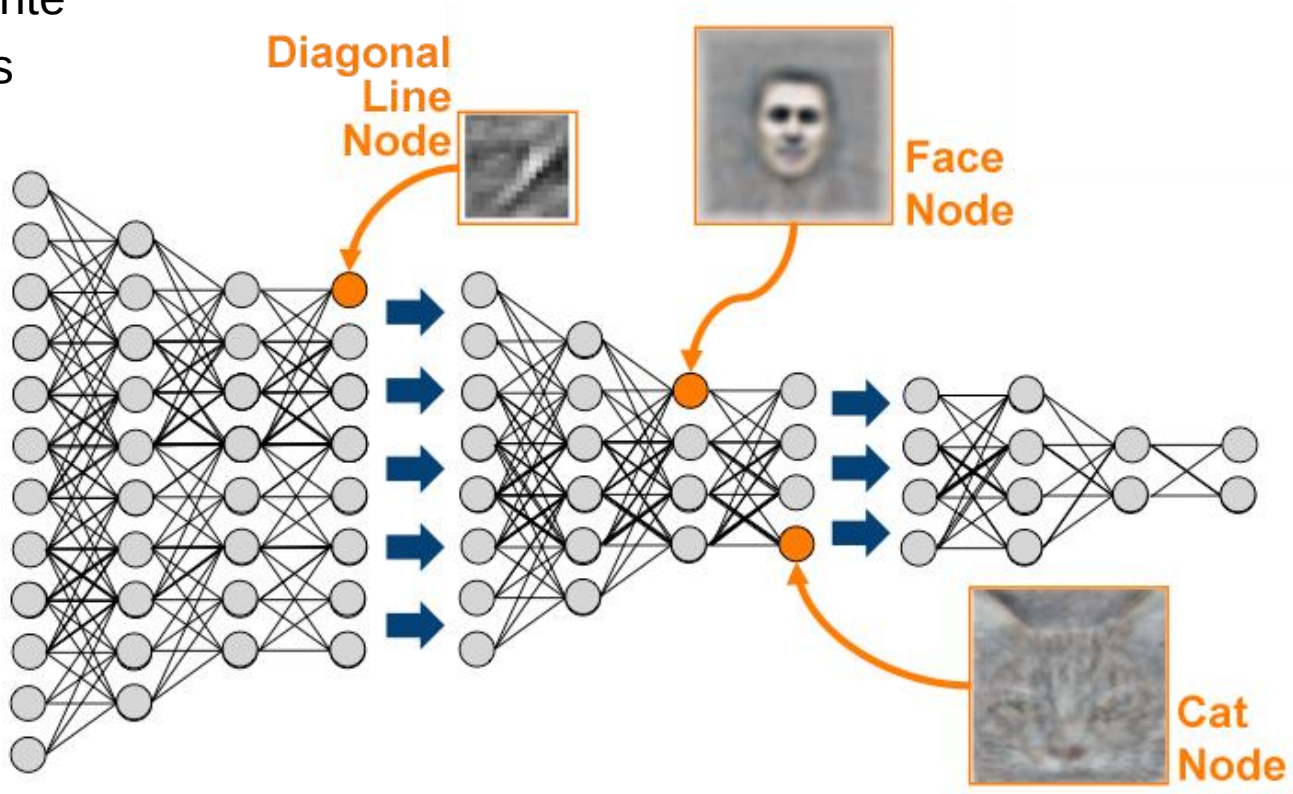


- AudiCup



Deep Learning
Beispiel Google

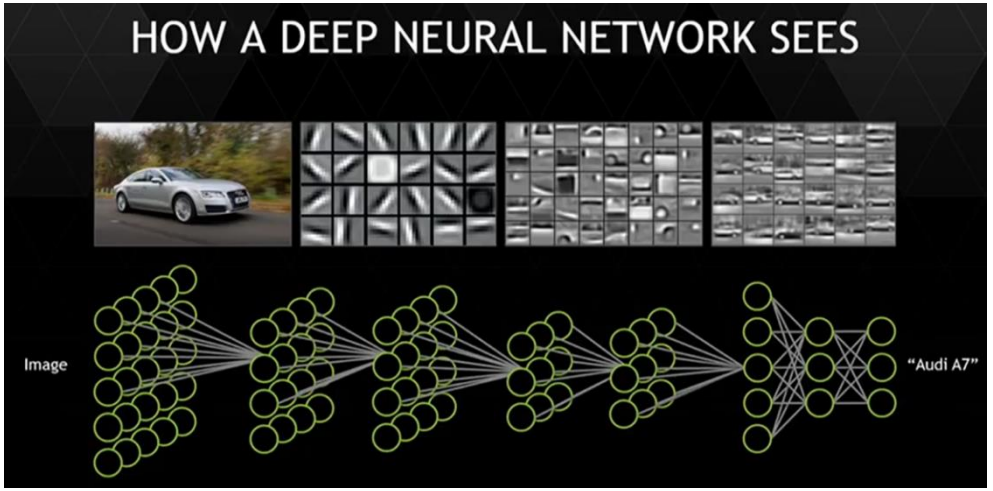
- Input
  - 10 Mio Bilder (200 x 200 pixel)
- Learning
  - 1 Mio Gewichte
  - 16.000 cores
  - 3 Tage



http://theanalyticsstore.com/deep-learning/

Deep Learning
Beispiel NVIDIA

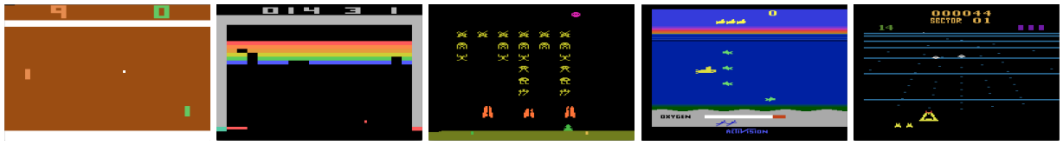
- Beispiel NVIDIA
  - erste Schicht erkennt Linien und Kreise
  - Fahrzeugteile
  - Fahrzeuge
  - Fahrzeugtyp
- Aufwand Lernen
  - Tage
    (auf GPU Cluster)
- Leistung Recall
  - 2 Megapixel
  - 30 fps
  - 75 Objects



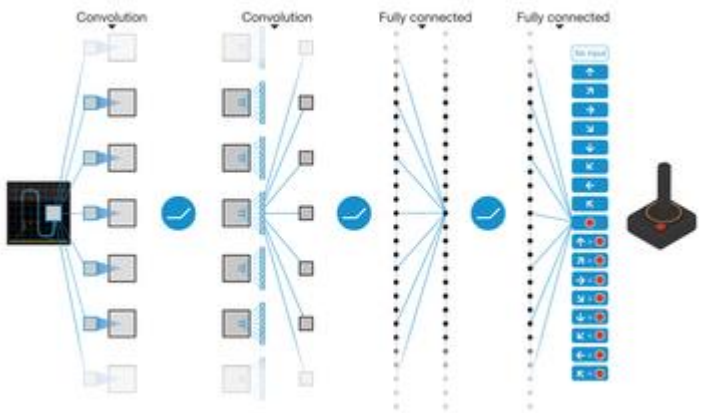
https://www.youtube.com/watch?v=zsVsUvx8ieo

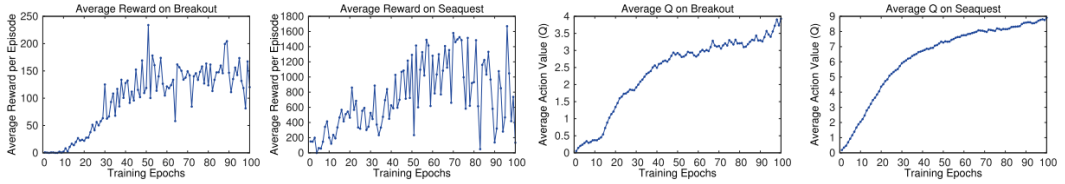
Deep Learning
Beispiel DeepMind

- Deep Reinforcement Learning von Computer Spielen



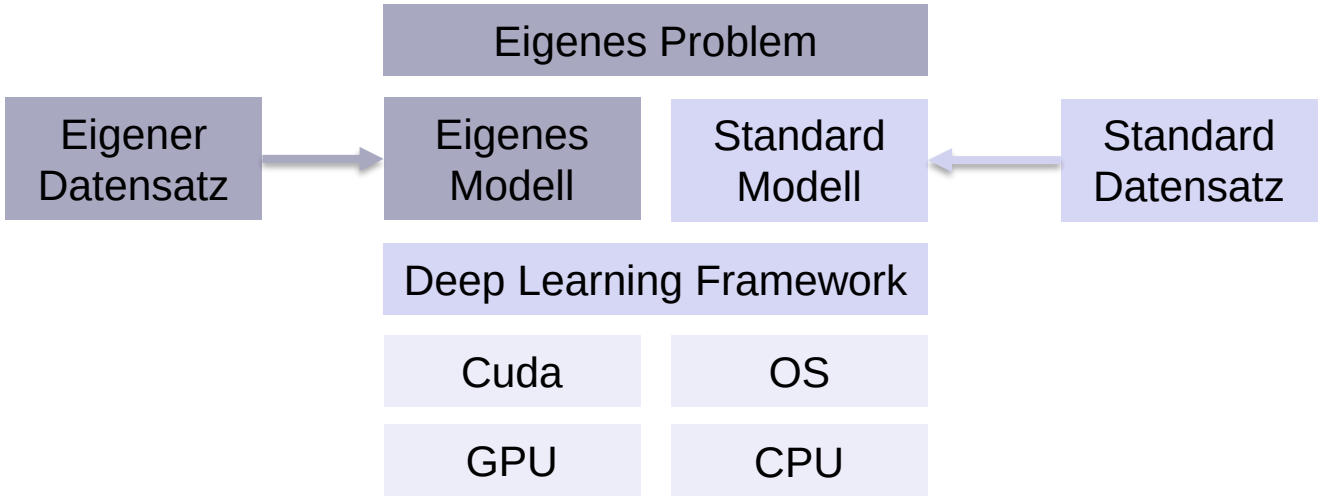
- Input
  - 84*84*4 downsampled live Video Input
- Netzwerk
  - 4 Layers, 2 Convolutional (8x8, 4x4), 2 fully connected
- Ergebnis
  - 4 der 7 Spiele besser als Human Expert



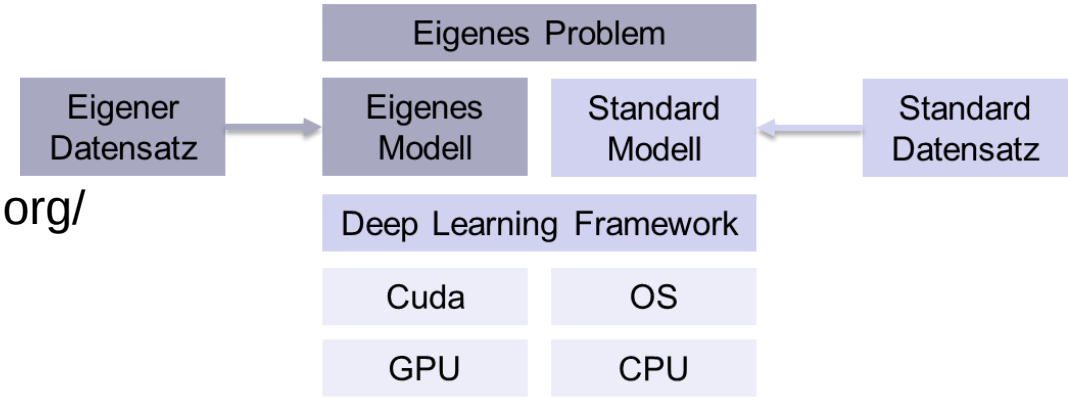


https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf und http://www.nature.com/nature/journal/v518/n7540/full/nature14236.html

Deep Learning
Frameworks



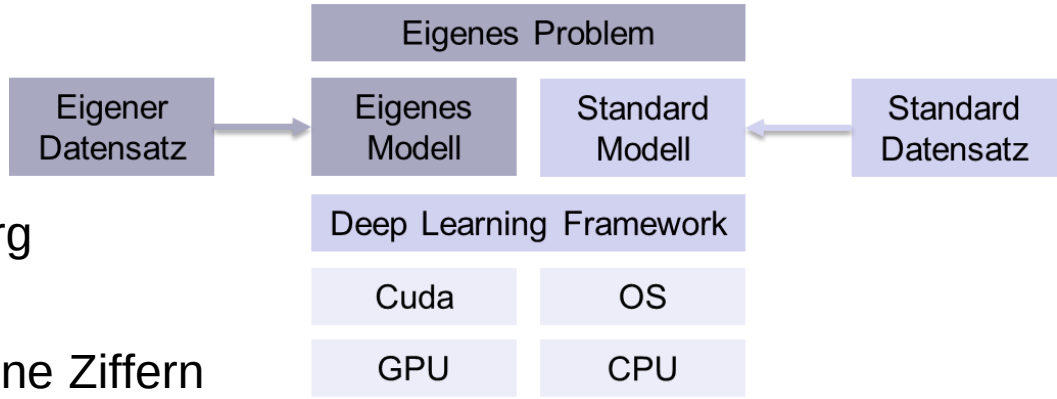
Deep Learning
Bekannte Frameworks



- TensorFlow
  - Google Brain team
  - https://www.tensorflow.org/
- Torch, PyTorch
  - Communities
  - http://pytorch.org/
- Deeplearning4j
  - Skymind engineering team, Deeplearning4j community
  - https://deeplearning4j.org/
- Caffe
  - Berkeley Vision and Learning Center
  - http://caffe.berkeleyvision.org/
- Caffe2
  - Facebook
  - https://research.fb.com/downloads/caffe2/

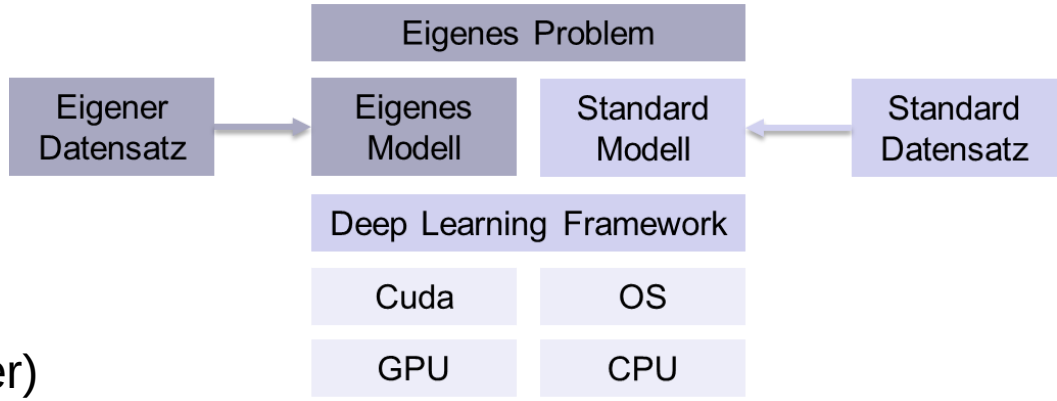
Deep Learning
Standard Datensätze

- ImageNet
  - 14 Mio getaggte Bilder
  - 21.000 Kategorien
  - http://www.image-net.org
- MNIST
  - 70.000 handgeschriebene Ziffern
  - http://yann.lecun.com/exdb/mnist/
- COCO (common objects in context)
  - 200.000 getaggte und segmentierte Bilder
  - http://cocodataset.org/#home
- Musik, Gesichter, Sprache, Texte, …



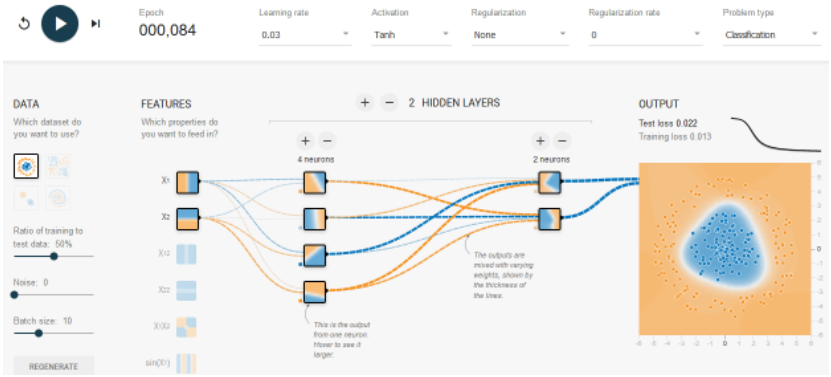
Deep Learning
Standard Modelle

- LeNet-5 (ab 1990)
  - 5 Schichten (4,1)
  - MNIST
- AlexNet (2012)
  - 8 Schichten (5,3)
  - ImageNet (16.4% Fehler)
- GoogLeNet (2014)
  - 22 Schichten
  - ImageNet (6.7%)
- ResNet-152 (2015)
  - 152 Schichten
  - ImageNet (3.6%), COCO
- VGGNet, Mobilenet, Inception, …

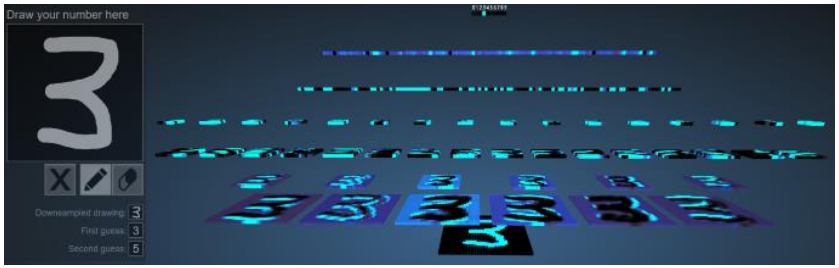


Deep Learning
Visualisierungen

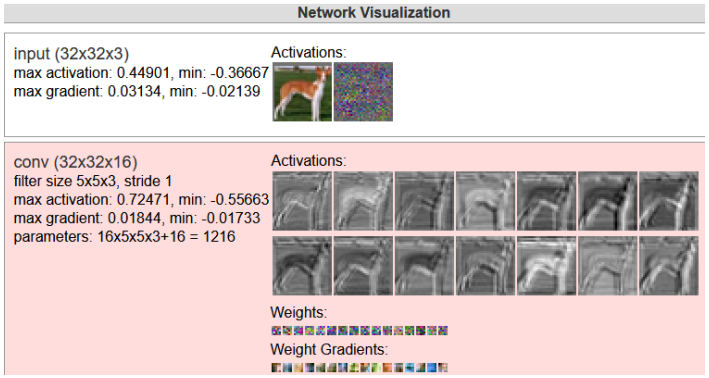
- Google Playground
  - http://playground.tensorflow.org



- 3D Ziffernerkennung (Adam Harley)
  - http://scs.ryerson.ca/~aharley/vis/conv

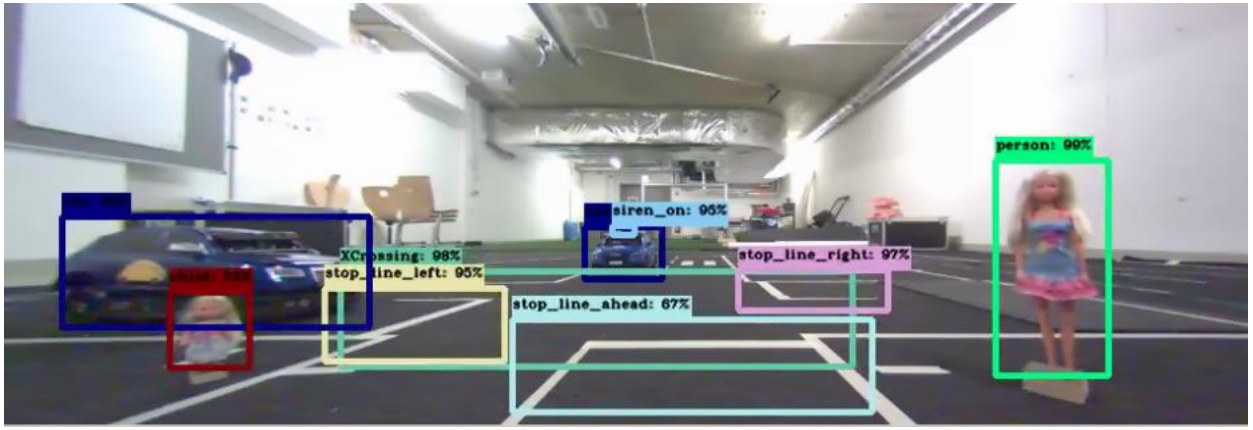


- ConvnetJS (Andrej Karpathy)
  - https://cs.stanford.edu/people/karpathy/convnetjs



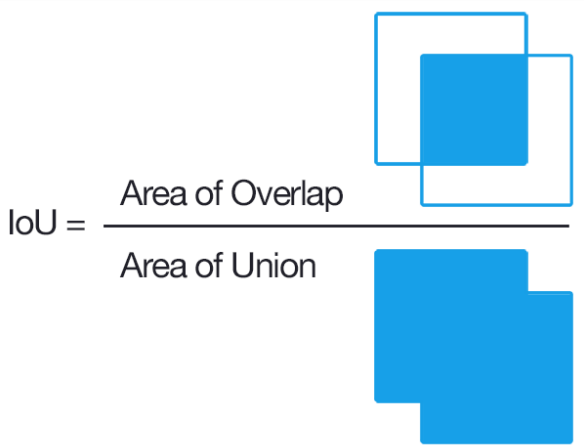
Objektlokalisierung

- Häufig sollen Objekte in Bildern nicht nur klassifiziert werden, sondern mehrere Objekte gleichzeitig erkannt und auf dem Bild verortet werden
  - Ergebnis ist eine Liste von Bounding Boxes mit Kategorie und confidence
- Two shot detectors
  - Erster Schritt: region proposals
  - Zweiter Schritt: Klassifizierung der Regionskandidaten
  - R-CNN Familie
- Single Shot detectors
  - Lokalisierung und Klassifizierung in einem einzigen forward pass
  - YOLOv1-3, SSD Multibox, RetinaNet
- Beispiel
  - Faster R-CNN (region props)
  - Inception v2 (detector)



Objektlokalisierung

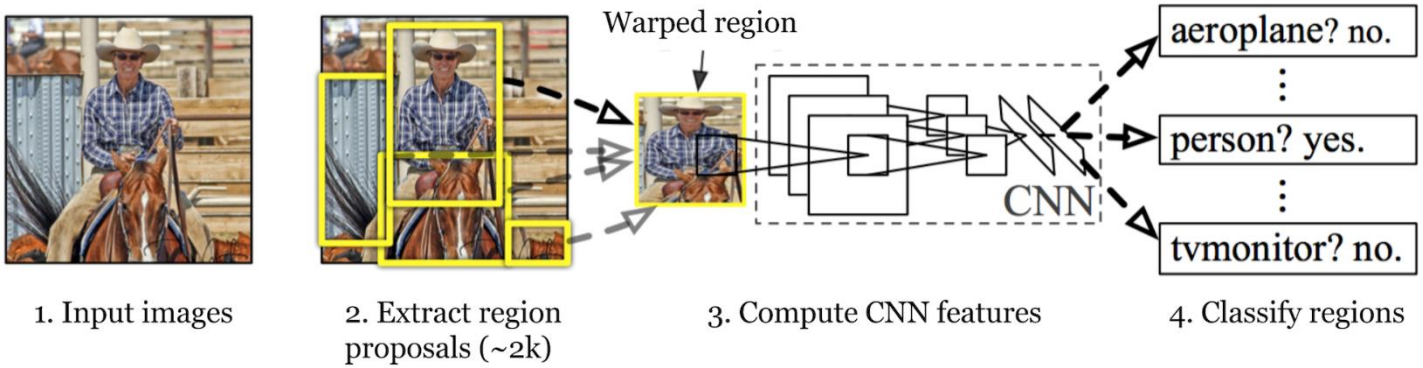
- Evaluationsmetriken
  - $Precision = \frac{TP}{TP+FP}$
  - $Recall = \frac{TP}{TP+FN}$
  - IoU: Intersection over Union
  - AP@0.5: average precision with IoU = 0.5
  - AP^{small}: AP kleiner Objekte (<32^2 Pixel) (medium < 96^2) (large)
  - mAP: Mittelwert über alle Klassen



https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Objektlokalisierung

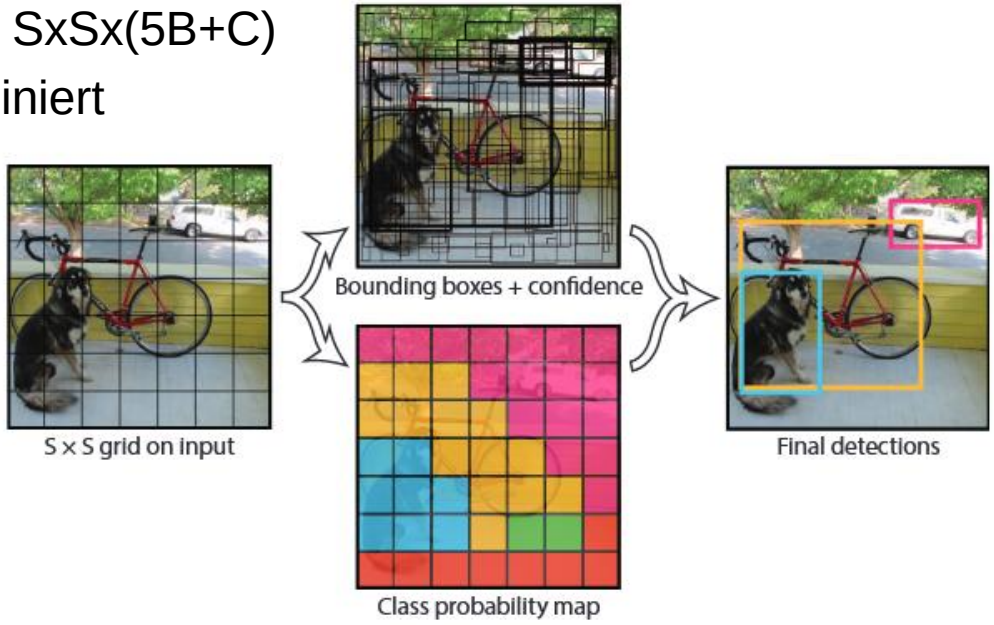
- Two shot detectors (am Beispiel R-CNN)
  - Schritt 1: Regionskandidaten finden (durch selective search, ca 2000)
  - Schritt 2: Jede Region durch CNN klassifizieren
    - Region auf feste Größe skalieren für CNN
    - Klassifizierung durch CNN
    - Non-max suppression eliminiert redundante Boxen
- Nachteil
  - Langsam
  - Einzelne Komponenten müssen getrennt gelernt werden



Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik (2013) Rich feature hierarchies for accurate object detection and semantic segmentation

Objektlokalisierung

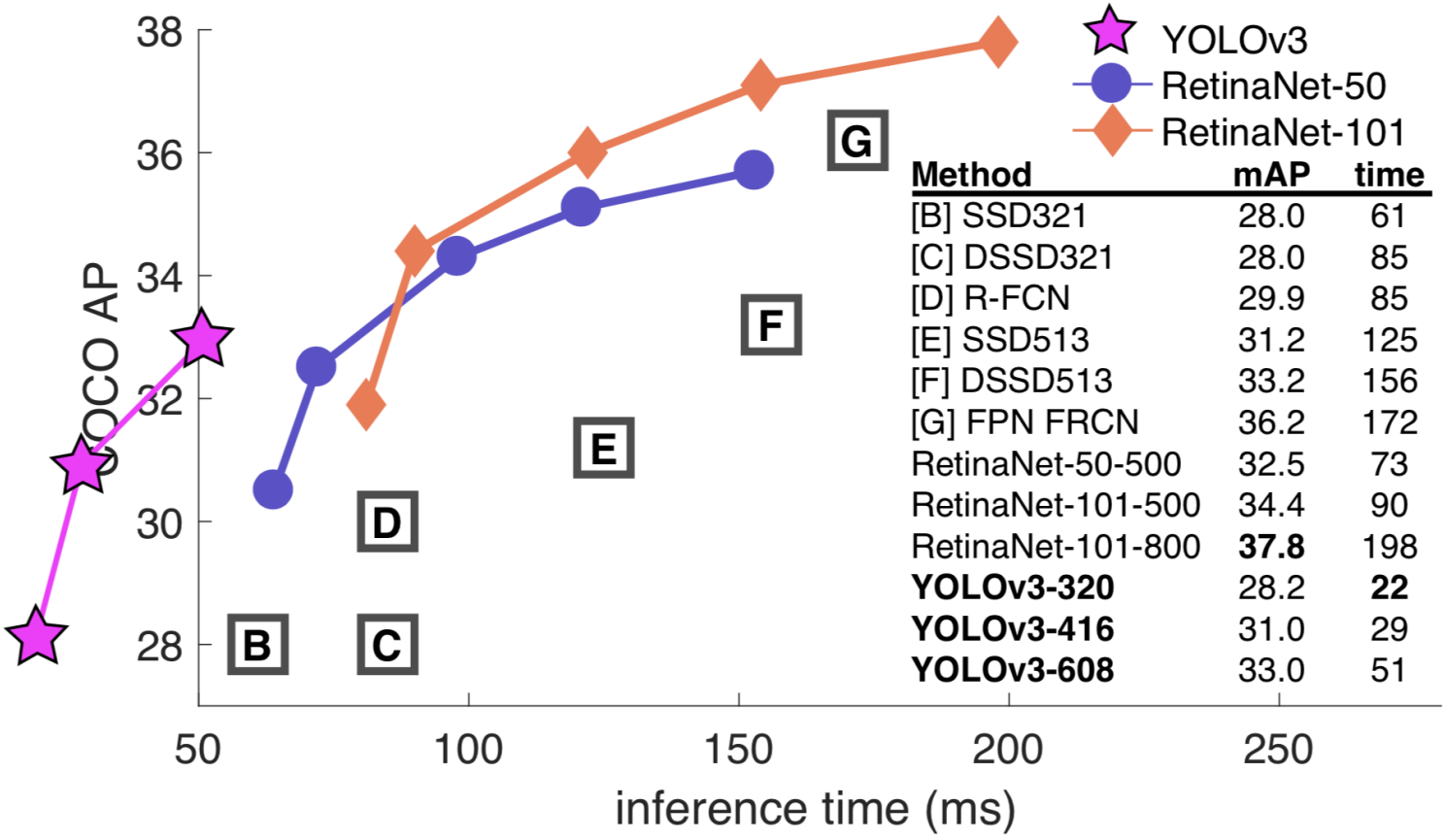
- Single shot detectors (am Beispiel YOLO)
  - CNN wird für Objekterkennung vortrainiert
  - Bild wird in SxS Zellen unterteilt
  - Jede Zelle bekommt B bounding boxes (bei YOLO B=2)
    - x, y, w, h, c (confidence object in box)
  - Für jede Zelle wird für jede der C Klassen einen score (C=20 bei Pascal)
  - Output ist dann ein Tensor SxSx(5B+C)
  - Non-max suppression eliminiert redundante Boxen
  - Loss ist Summe aus
    - Localization loss
    - Classification loss



Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi (2015) You Only Look Once:Unified, Real-Time Object Detection

Objektlokalisierung

- Vergleich verschiedener SSD Verfahren



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

https://medium.com/@jonathan_hui/object-detection-speed-and-accuracy-comparison-faster-r-cnn-r-fcn-ssd-and-yolo-5425656ae359

Neuronale Netzwerke

- Generalisierung
  - Gute Generalisierung bei vielen Funktionen
  - Meist dann, wenn die Inputs relativ unabhängig sind und die Outputs gleichmäßig mit den Inputs variieren
- Rauschen
  - Sehr tolerant gegenüber Rauschen in den Input Daten
- Transparenz
  - Netzwerk ist eine Black Box, es gibt keine Erklärung, warum es zu dem berechneten Output kam
- Vorwissen
  - Netzwerk ist eine Black Box, Vorwissen kann nicht (einfach) vorgegeben werden

Anwendungen

- Handschrifterkennung für Postleitzahlen
  - LeCun 1989
  - 16*16 array input, 3 hidden layers (768, 192, 30 units) nicht vollständig vernetzt, 10 output units
- Fahren auf der Autobahn
  - ALVINN, Pomerlau 1993
  - 30*32 video input, 5 hidden units, 30 output units
- Deep Learning
  - Google 2012, Nvidia 2014, DeepMind 2015
  - Bilderkennung, Autonomes Fahren
  - Sprachverstehen (Sprecherunabhängig)
  - Textanalyse, Predictive Maintenance, …
- …