Karlsruher Institut für Technologie

Maschinelles Lernen II - Fortgeschrittene Verfahren
V04 Deep Learning – Deep Belief Networks

Sommersemester 2017
Prof. Dr. J.M. Zöllner, Prof. Dr. R. Dillmann

Nach Geoffrey E. Hinton
(derzeit Uni. Toronto & Google)

INSTITUT FÜR ANGEWANDTE INFORMATIK UND FORMALE BESCHREIBUNGSVERFAHREN
INSTITUT FÜR ANTHROPOMATIK UND ROBOTIK

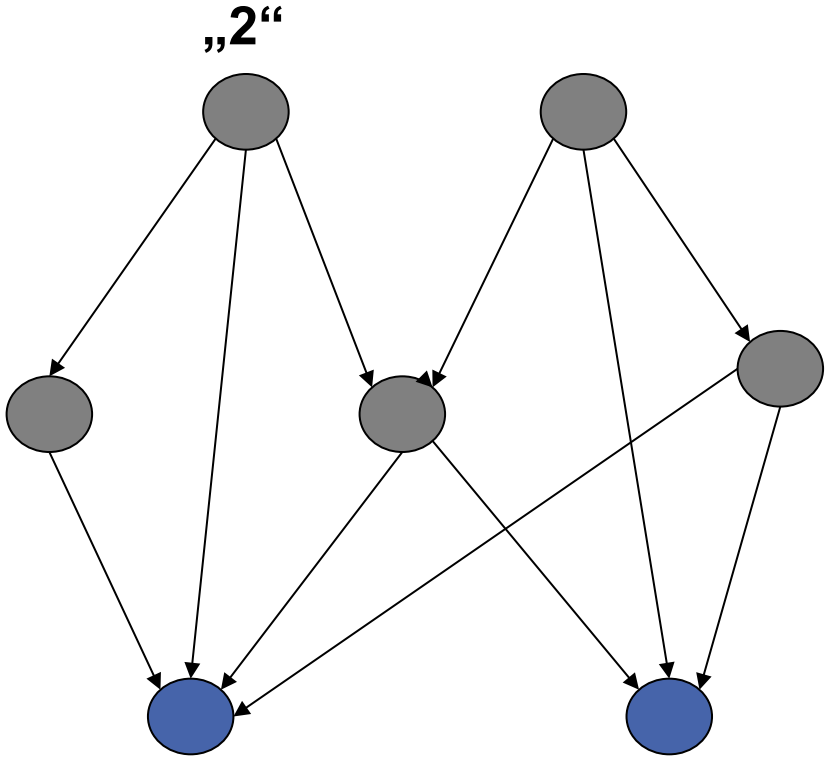
Übersicht

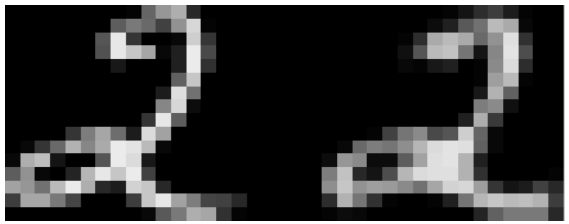
- Motivation
  - Belief Netze
  - ➔ Nutzen
  - ➔ Prinzip
  - ➔ Problematik des Lernens
- Restricted Boltzmann Maschinen (RBM)
- Tiefe Netze mit RBM (DBM)
- Äquivalenz zu Sigmoiden Belief Netzen
- Anwendungsmöglichkeiten

Die Vision ....

- Es gibt ein „Netz“
- das sieht Daten
- kann sie „erklären“
- und sie erzeugen
- und das Netz lernt dies auch (weitgehend) selbst (unüberwacht oder zumindest semi-überwacht) ?

→ Deep Learning geht in diese Richtung ☺





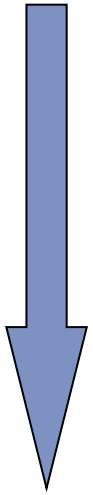
Motivation (zum Durchhalten ☺)

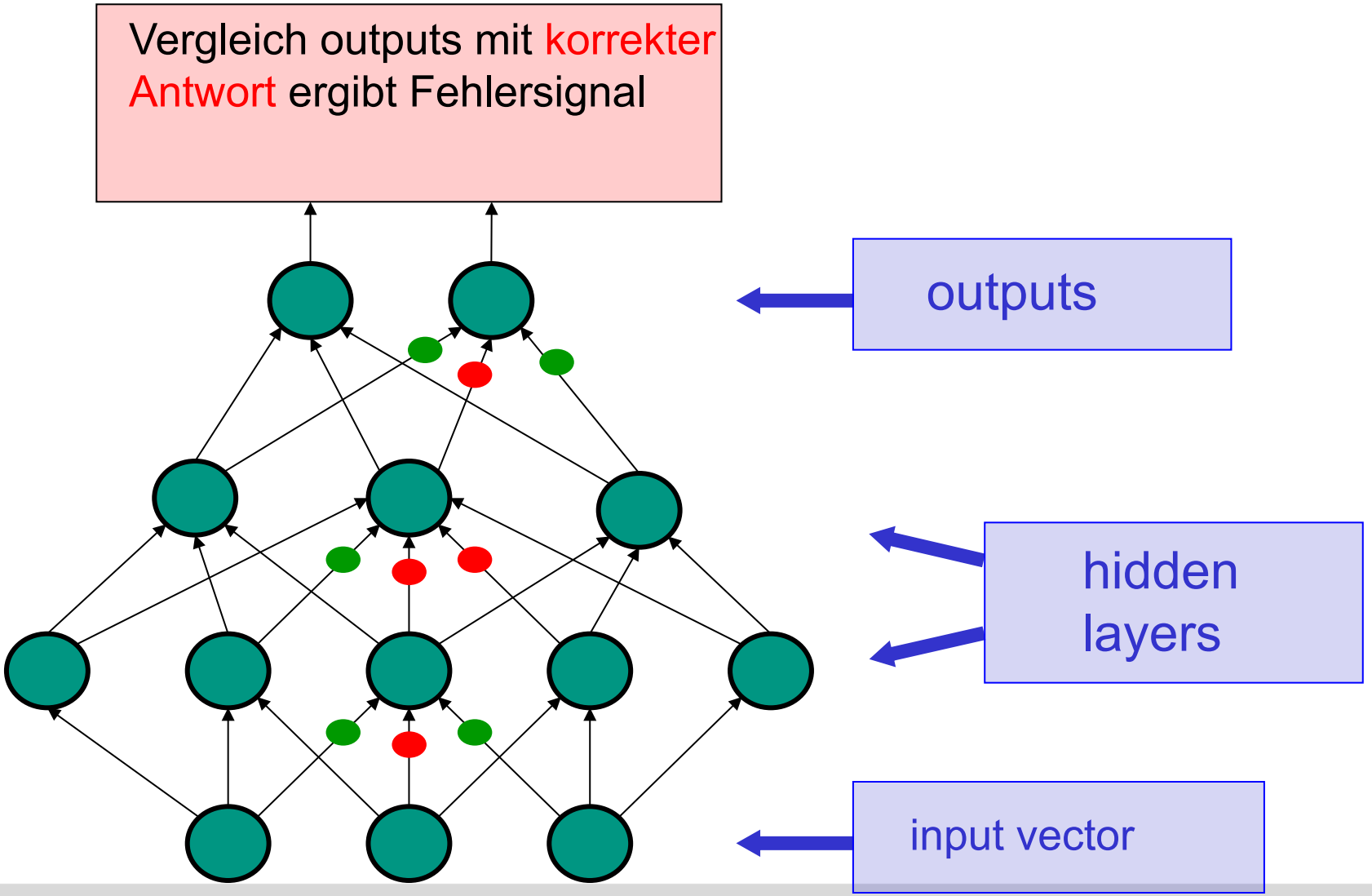
Videos von Netzen die Ziffern generieren

➔ http://www.cs.toronto.edu/~hinton/adi/index.htm

2. Generation Neuronaler Netzte (~1985) hat es bereits versucht

Rück-Propagierung (back-propagation) des Fehlers ➔ Gradient fürs Lernen





Dann: “A temporary digression”

- Vapnik und Kollegen entwickeln die “very clever type of perceptron”[Hinton] - Support Vector Machine
  - “a clever optimization technique is used”
  - “But its just a perceptron and has all the same limitations”

- Hinton: „In the 1990’s, many researchers abandoned neural networks with multiple adaptive hidden layers because Support Vector Machines worked better.“

Was war „falsch“ an Backpropagation?

- Überwachtes Lernen: Benötig zu viele Trainingsdaten mit Zielwerten (labeled)
  - Aber die meisten realen Daten sind ohne Zielwerte (unlabeled)
- Performanz skaliert oft nicht wirklich gut
  - Sehr langsames Lernen (Konvergenz) in Netzen mit vielen Hidden Layer ← diese werden leider benötig
- Lokale Minima
- Overfitting ....

Was war gut an Backpropagation

- Effizienz und Einfachheit (z.B. Gradienten-Abstieg für das Backprop Lernen)
  ➔Ziel: beibehalten!

Aber

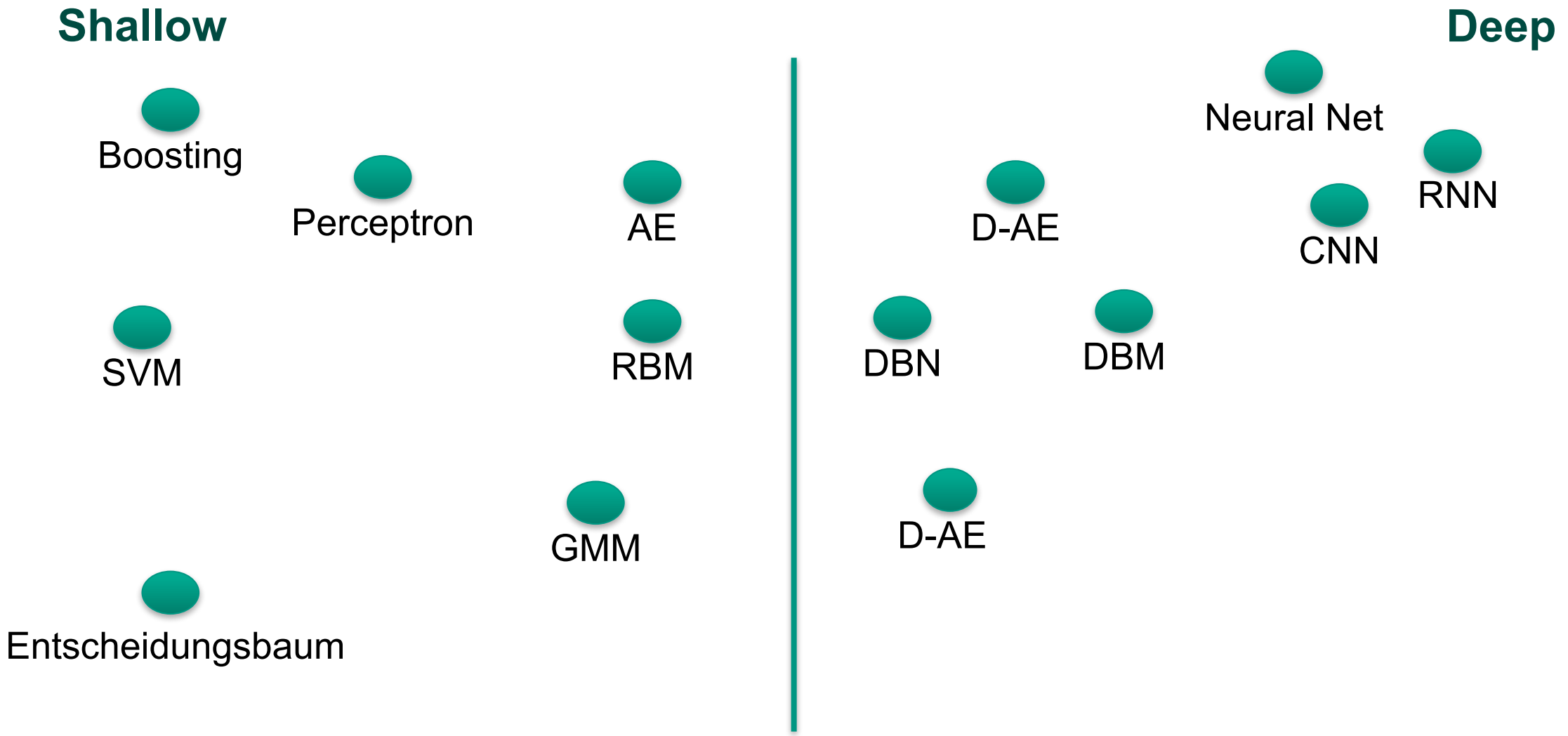
- Die Struktur der (Input) Daten ist wichtig und sollte gelernt werden
  ➔ Ziel: Generatives Modell nutzen! (siehe auch SSL)
  - Anpassen z.B. der Gewichte eines Netzes, so dass die Wahrscheinlichkeit, dass das Netz die Daten produziert maximiert wird
  - Z.B.: Ansatz bei Bildklassifikation = Netz finden, dass intern $p(image \mid w)$ maximiert, nicht nur $p(label \mid image)$

- Deep Learning: Wie soll generativer Ansatz genutzt sein?
  Wie kann gelernt werden?
  Wie sieht Inferenz aus?

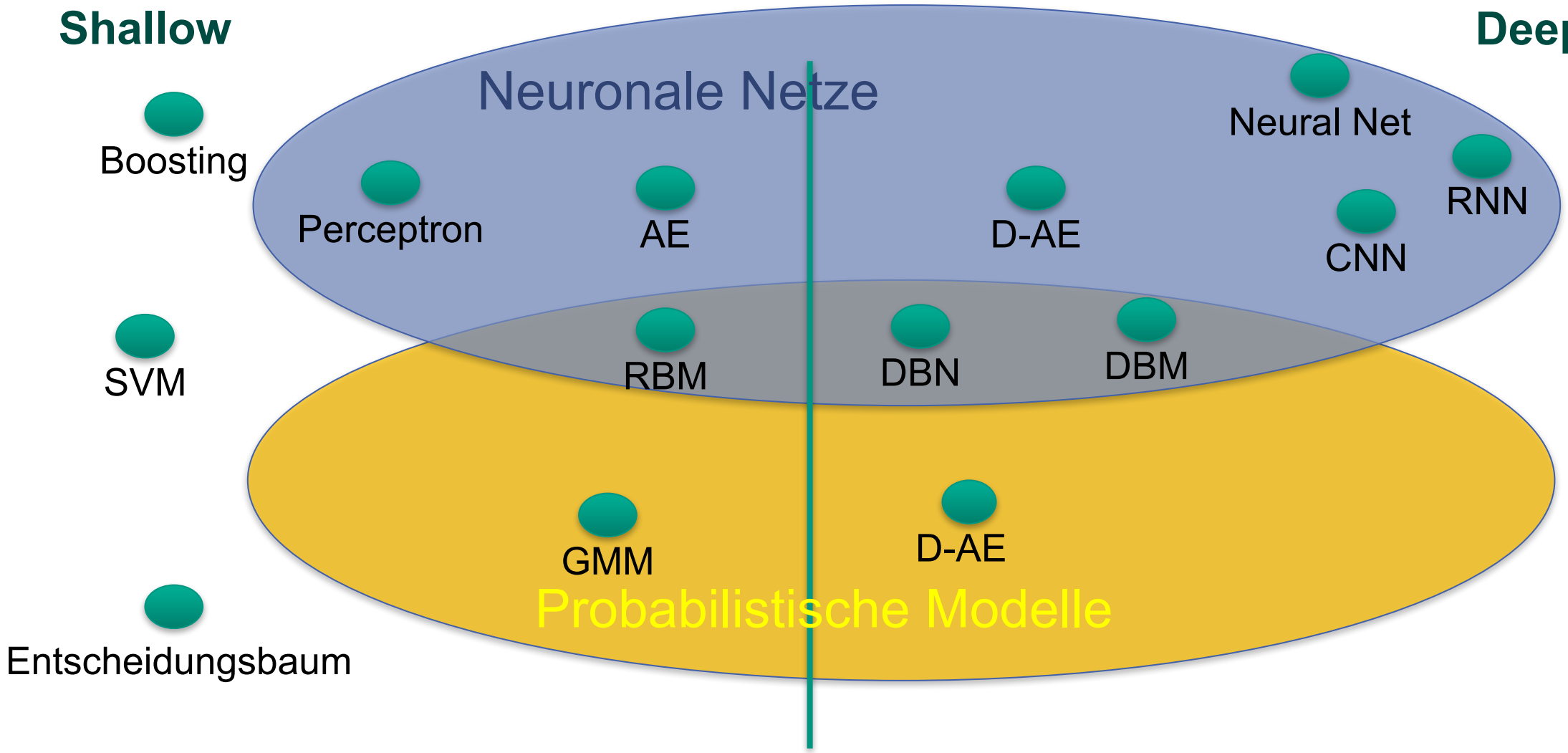
Wann sind Netze/Lernverfahren tief „deep“

- Hinton
  - „It is deep, if it contains at least two layers with non-linear transformations“
- LeCun (additional)
  - „It is deep, if it represents a feature hierarchy.“

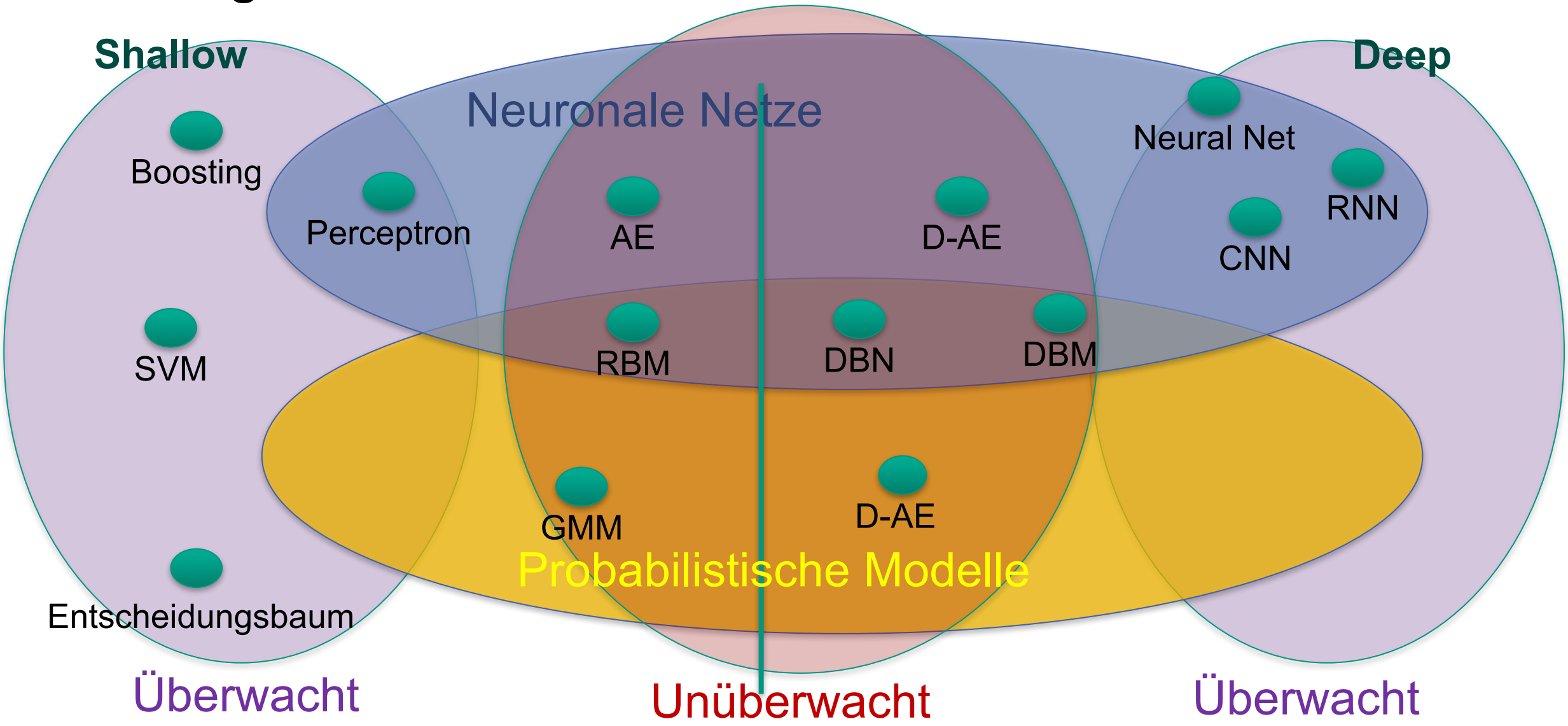
Einteilung Lernverfahren



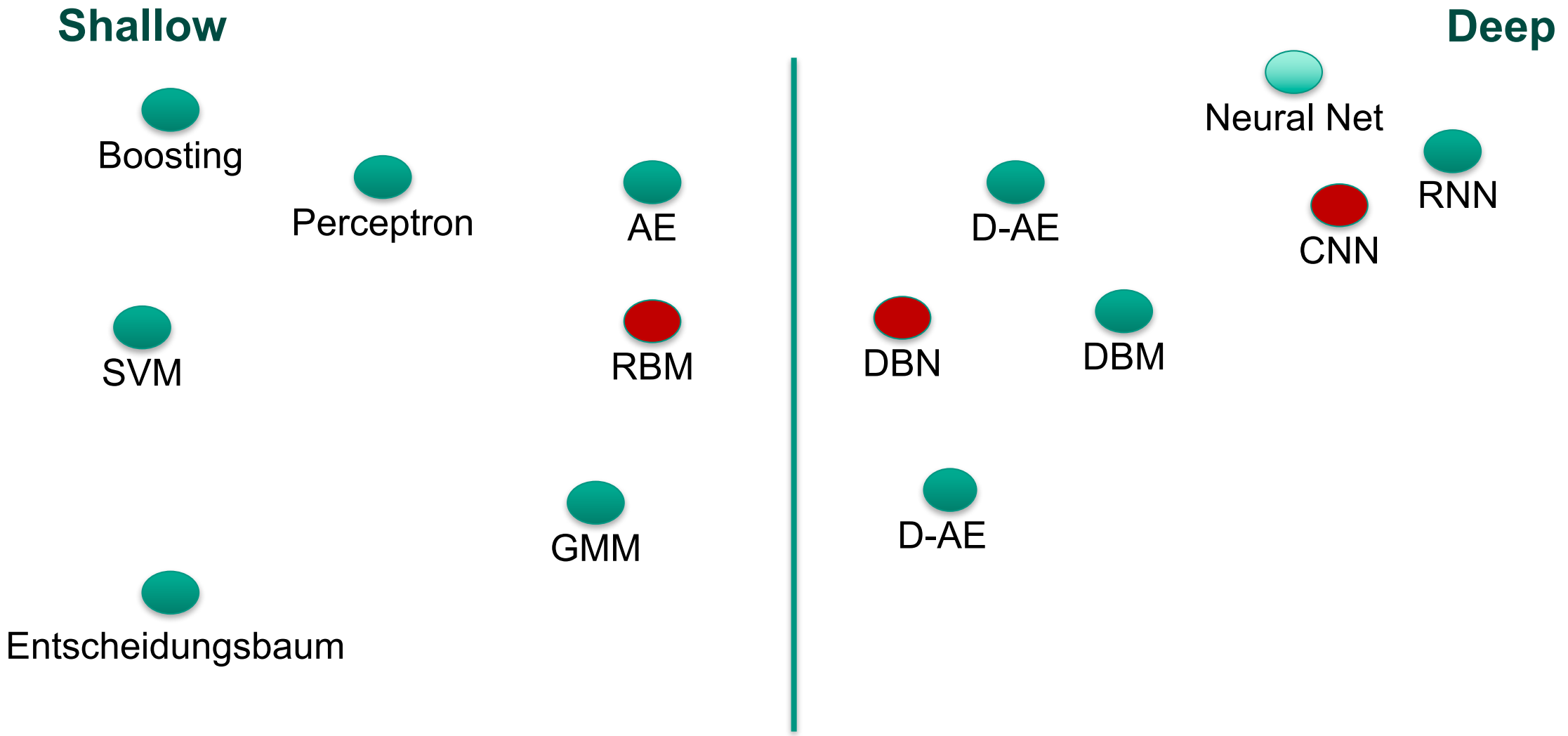
Einteilung Lernverfahren



Einteilung Lernverfahren



Fokus in der ML 2 Vorlesung



Deep Belief Netz

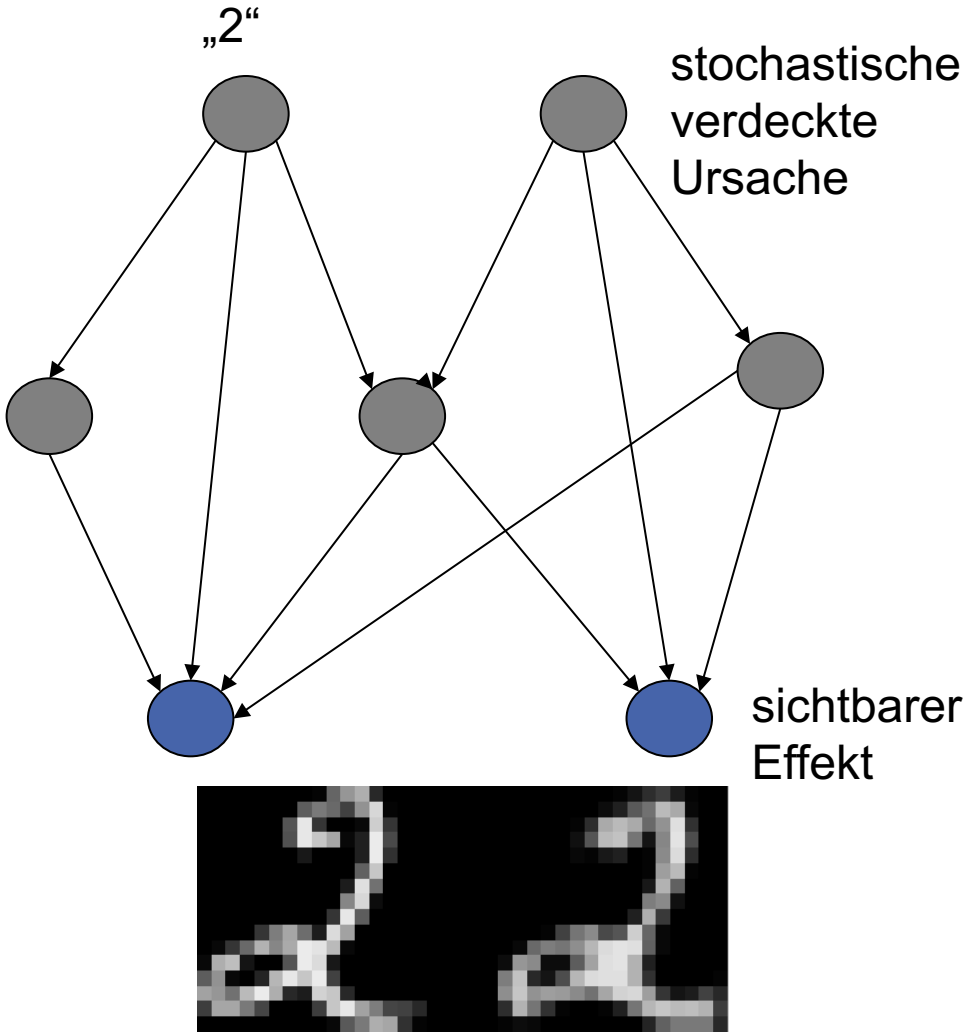
- Es ist ein „Netz“
- das sieht Daten
- und kann sie „erklären“

➔die (stochastischen) nicht beobachtbaren Ursachen der Daten können inferiert werden

- und erzeugen

➔das Netz kann sichtbaren Effekt erzeugen

- das Netz soll dies auch (weitgehend) selbst lernen (unüberwacht oder zumindest semi-überwacht)

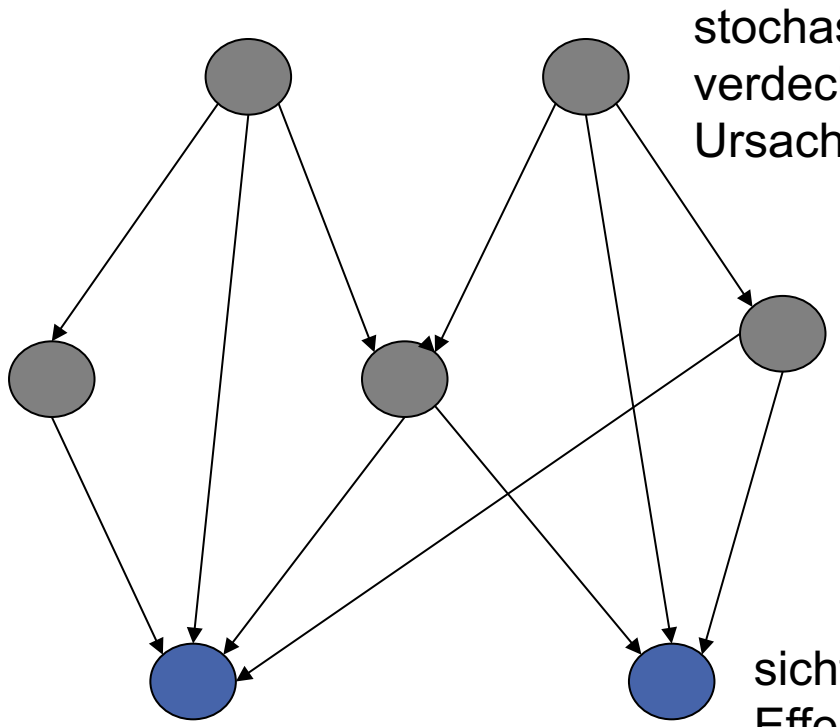


(Parametrisiertes) Belief Netz = korrektes Modell

- Ein Belief Netz ist ein gerichteter azyklischer Graph mit stochastischen Variablen. (graphisches probabilistisches Modell)
- Einige Variablen sind beobachtbar (Evidenzen, Daten)

Zwei Fragestellungen:

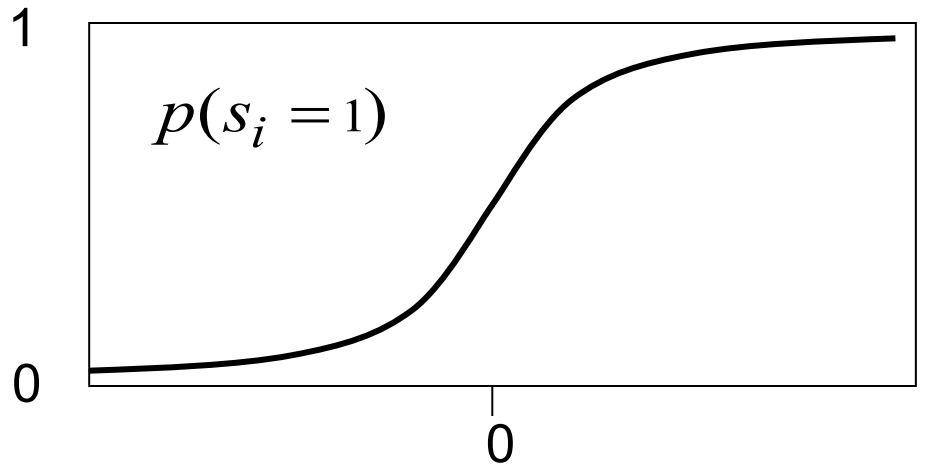
- Das Inferenz-Problem: Schließen auf den Zustand der verdeckten Variablen
- Das Lern-Problem: Anpassen der Verbindungen zwischen den Variablen um das Netz zu befähigen beobachtete Daten generieren zu können



Netz bestehend aus Schichten von binären stochastischen Variablen die gewichtet verbunden sind (Generalisierung auf andere Arten von Variablen möglich)

Einfaches Netz mit stochastische binären Einheiten (Bernoulli Variablen)

- Mögliche Zustände der Knoten: 1 oder 0



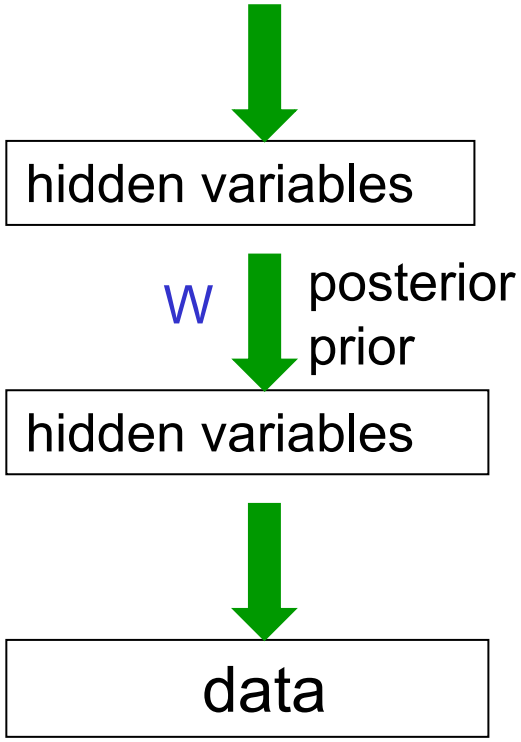
- Aktivierungswahrscheinlichkeit
  - bestimmt durch den gewichteten Input von anderen (Vorgänger) Einheiten (plus Bias)

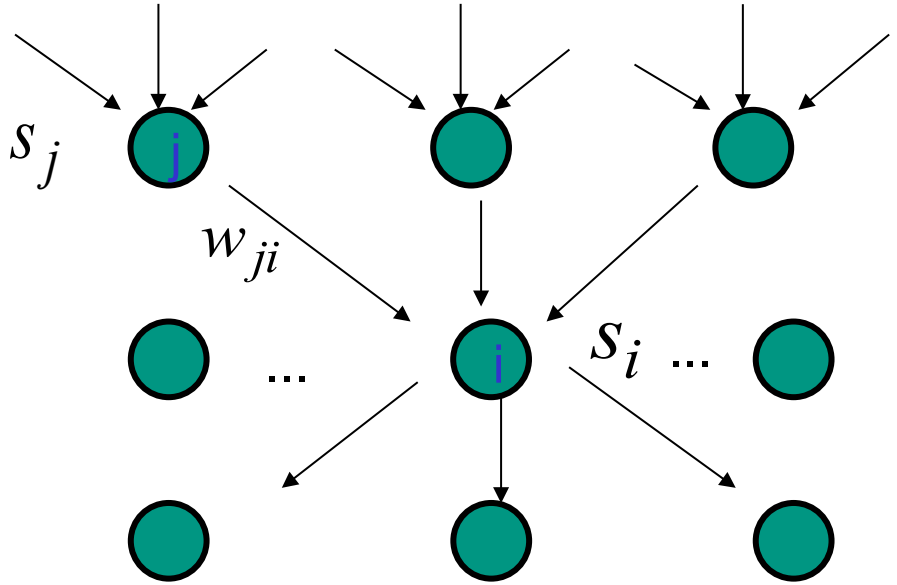
$$b_i + \sum_j s_j w_{ji}$$

  - z.B. für sigmoide Einheiten entsprechend der Sigmoid Regel:

$$p(s_i = 1) = \frac{1}{1 + \exp(-b_i - \sum_j s_j w_{ji})}$$

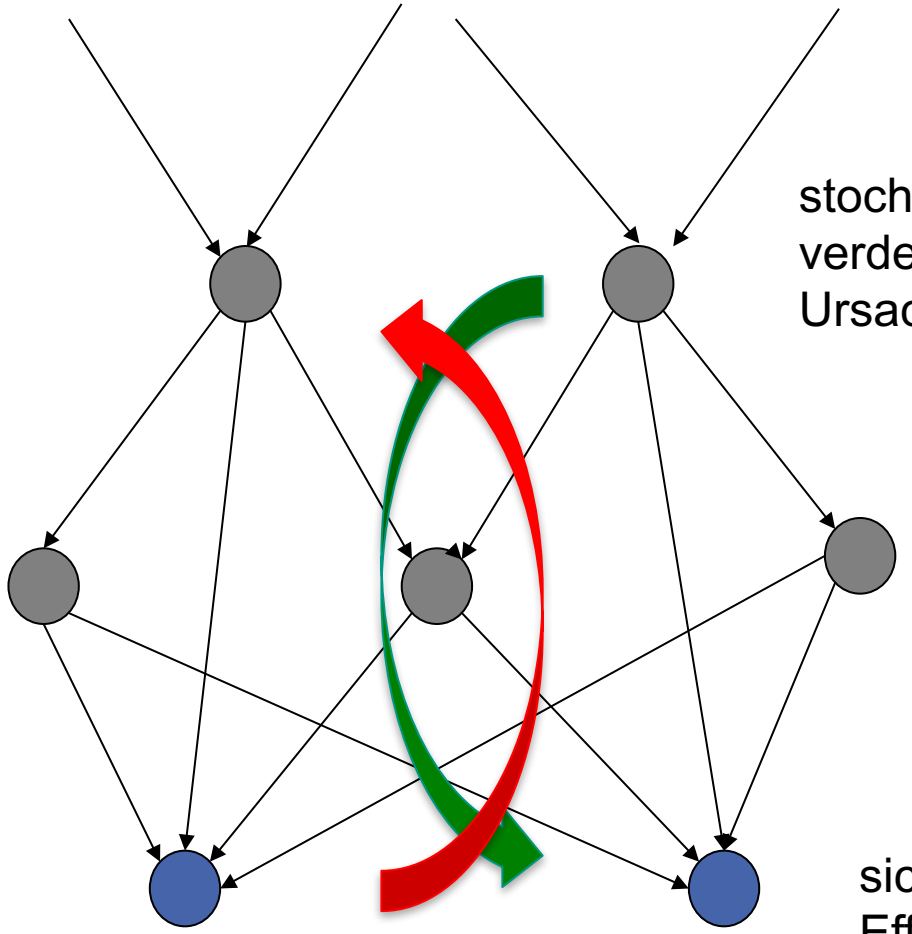
Darstellung (Erklärung)





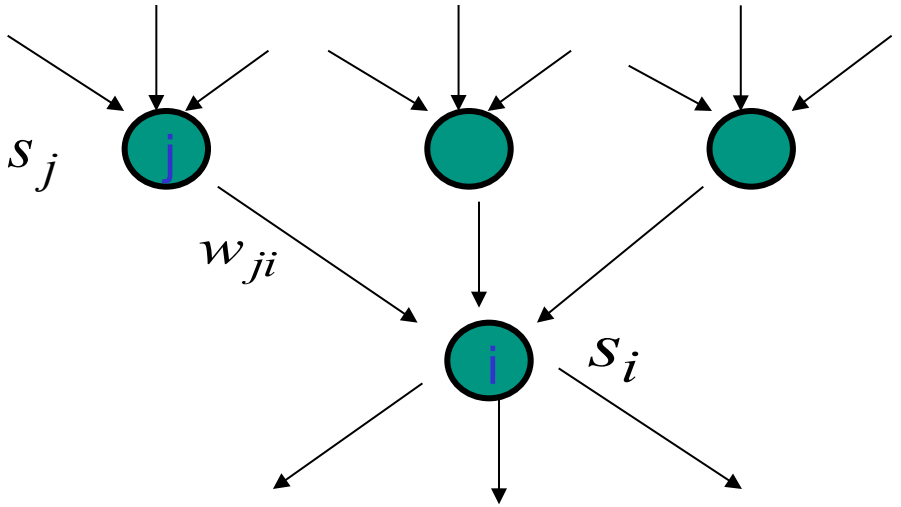
Lernen in „Deep“ Belief Nets

- Einfach (unbiased) Beispiele an den Endknoten zu generieren um zu sehen woran das Netz „glaubt“. (Belief Net ☺)
- Schwer die a-posteriori Wahrscheinlichkeit für alle Konfigurationen der verdeckten Ursachen zu inferieren
- Wie kann man dann in Deep Belief Netzen mit vielen Ebenen und Millionen von Parametern lernen?



Lernen für Sigmoide Belief Netze

- Lernen einfach wenn
  - Daten der verdeckten Zustände (samples der a-posteriori Verteilung) gegeben der beobachteten Daten vorliegen
- Idee des Lernens:
  - Für jede Einheit:
    Maximiere die Wahrscheinlichkeit, dass der beobachtete Zustand s_i von den binären Zuständen der Vorgänger generiert wird (Log Likelihood)
- Problem: Wenn überhaupt nur für unterstes Ebenen (mit sichtbaren Daten) möglich



$$p_i \equiv p(s_i = 1) = \frac{1}{1 + \exp(-\sum_j s_j w_{ji})}$$

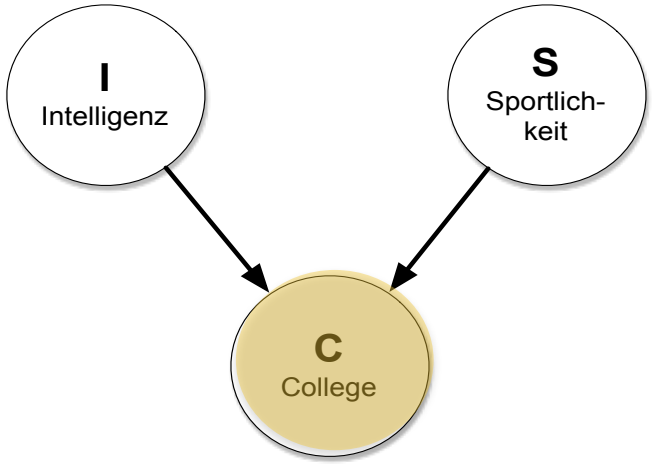
$$\Delta w_{ji} = \varepsilon \, s_j (s_i - p_i)$$

Lernrate

Möglichkeit: Inferenz verdeckter Zustände ←→Problem: Explaining away (Wdh.)

- Ergebnisse der Inferenz:

$$P(intelligent|college) \approx 0.71$$
$$P(intelligent|sportlich, college) \approx 0.63$$



➢ Die Beobachtung, dass jmd. auf ein College geht und er sportlich ist, lässt diesen automatisch weniger intelligent erscheinen

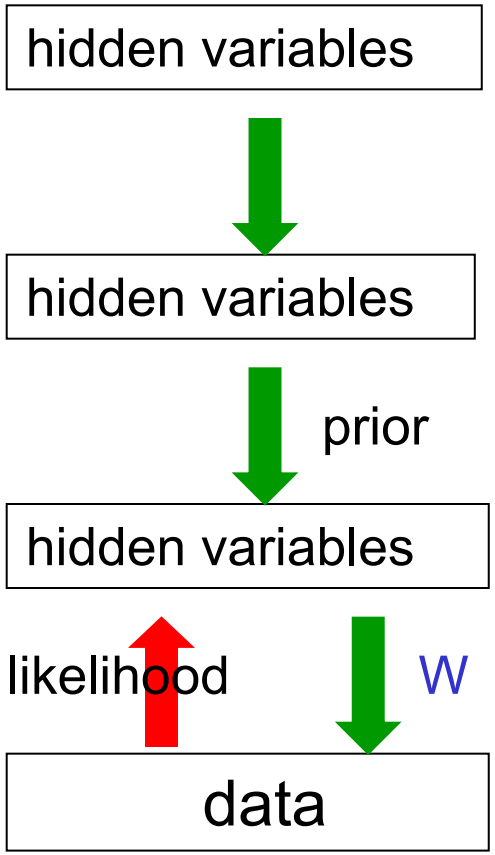
- Die Beobachtung einer Ursache, die ausreicht um einen Effekt zu erklären, lässt andere mögliche Ursachen automatisch unwahrscheinlicher werden
  - Weil die Beobachtung des gemeinsamen Effekts macht die Ursachen bedingt abhängig (Fall 3)
- Rück - Inferenz nicht möglich (durch Beobachtung der Effektes)

Probleme beim Lernen vielschichtiger sigmoid Belief Netze

- Lernen von W, benötigt die a-posteriori Verteilung des ersten hidden layer.

Probleme:

- Die a-posteriori Verteilung vorab durch beobachtete Daten zu inferieren ist problematisch wg. “explaining away”.
- Die a-posteriori Verteilung jeder Schicht hängt von der a-priori Verteilung ab
  →Um W zu lernen, benötigt man die Gewichte der höheren Schichten
- Wir müssten über alle möglichen Konfigurationen der höheren Variablen integrieren (Marginalisierung) um die a-priori Verteilung der ersten verdeckten Ebenen zu schätzen.
- ☹!



Übersicht

- Motivation
  - Belief Netze
  - ➔ Nutzen
  - ➔ Prinzip
  - Problematik des Lernens
- Restricted Boltzmann Maschinen (RBM)
- Tiefe Netze mit RBM (DBM)
- Äquivalenz zu Sigmoiden Belief Netzen
- Anwendungsmöglichkeiten

Idee: Einfacher lernbare Netze verwenden

Typen von generativen Netzen:

- Verbindung der binären stochastischen Einheiten in einem gerichteten azyklischen Graphen führt zu einem Sigmoid Belief Netz (Radford Neal 1992)

➔Problematisch

- Verbindung der binären stochastischen Einheiten mit **symmetrischen Verbindungen** führt zu einer Boltzmann Maschine (Hinton & Sejnowski, 1983).

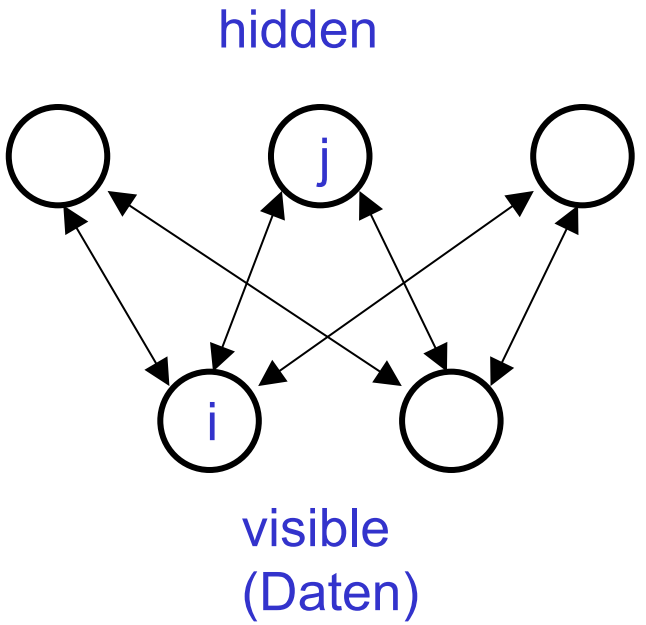
➔Bei Restriktion der Verbindung kann eine Boltzmann Maschine (einfach) gelernt werden.

Restricted Boltzmann Machine
(Smolensky ,1986, auch “harmoniums” genannt)

- Beschränkung der Verbindungen um lernen zu können:
  - Nur ein hidden layer
  - Keine Verbindung zwischen Einheiten gleicher layer

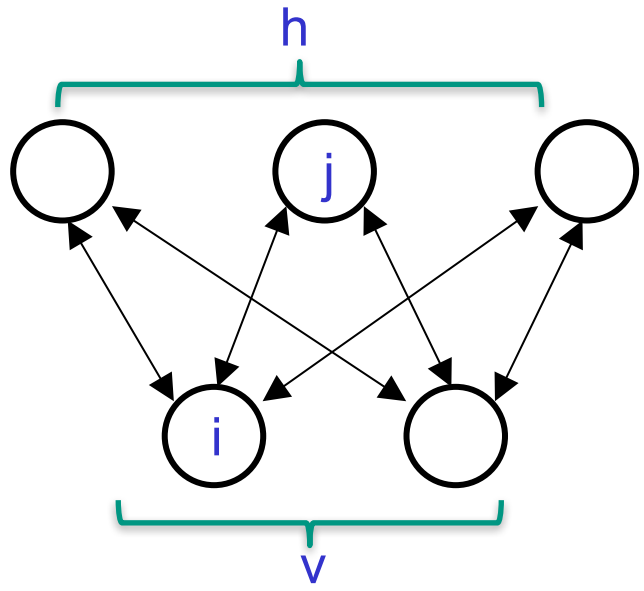
- In einem RBM, sind die hidden Einheiten unabhängig, wenn die beobachtbaren Variablen (Zustände) gegeben sind .
  - man erhält „einfach“ den Zustand (sample der a-posteriori Verteilung) der hidden Einheiten gegeben ein Daten-Vektor
    ← sigmoid Regel (aufwärts)
  - Wesentlicher Vorteil gegenüber den gerichteten Belief Netzen

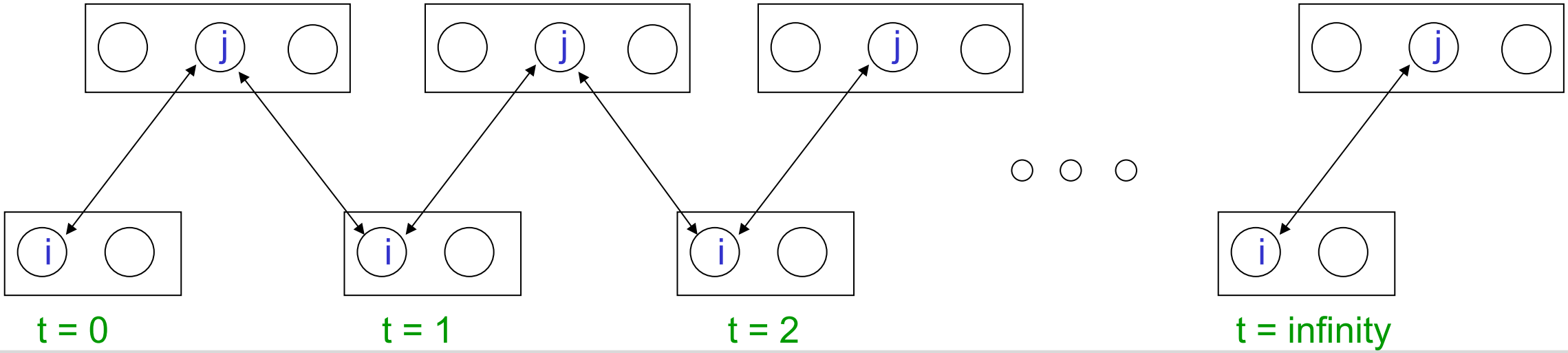
- Wie kann man Lernen?



Boltzmann Maschinen

- Arbeitsprinzip
  - Anstoßen (Erregen)
  - Iterieren in stabile Zustände (d.h. Aktivierungen der Knoten)
  - Viele solcher Zustände existieren

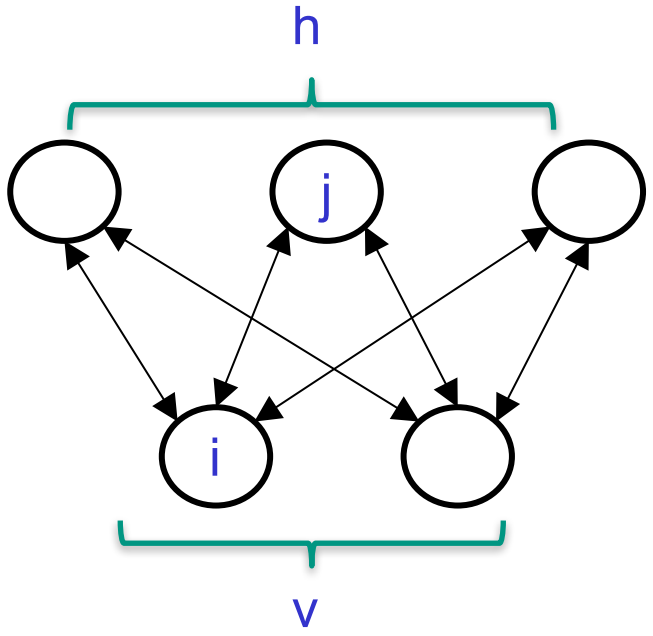




Boltzmann Maschinen - Lernen

- Ziel: Erzeugen stabiler Zustände die die sichtbaren Lerndaten D abbilden

- Prinzip: Maximum likelihood Lernen ➔ Log - Wahrscheinlichkeit für Zustände (Konfigurationen) welche die Lerndaten in den sichtbaren Einheiten enthalten maximieren



$$\log L = \sum_{\mathbf{v} \in \mathcal{D}} \log p(\mathbf{v})$$

- Methode: Betrachten der s.g. Energien der Konfigurationen

Definition: Energie einer Konfiguration
(Bias wird hier ignoriert)

Binärer Zustand der sichtbaren Einheit i

Binärer Zustand der hidden Einheit j

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij}$$

Energie mit Konfiguration
v an sichtbaren Einheiten
h an unsichtbaren Einheiten

Gewicht zwischen Einheiten i und j

$$-\frac{\partial E(v,h)}{\partial w_{ij}} = v_i h_j$$

Gewicht→ Energie → Wahrscheinlichkeit → Lernen

- Jede mögliche Konfiguration von sichtbaren und verdeckten Einheiten hat eine Energie
  → ist bestimmt durch Gewichte und Bias

- Die Energie einer gemeinsamen Konfiguration bestimmt deren Wahrscheinlichkeit (nächste Folie genauer) :

$$p(v,h) \propto e^{-E(v,h)}$$

$$p(v) \propto \sum_{h} e^{-E(v,h)}$$

→ soll maximiert werden = Lernen

Energien → Wahrscheinlichkeiten → Lernen

- Die Wahrscheinlichkeit einer gemeinsamen Konfiguration von sichtbaren und unsichtbaren Einheiten

  → hängt ab von der Energie einer Konfiguration in Vergleich zu allen anderen Konfigurationen.

$$p(v,h) = \frac{e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}$$

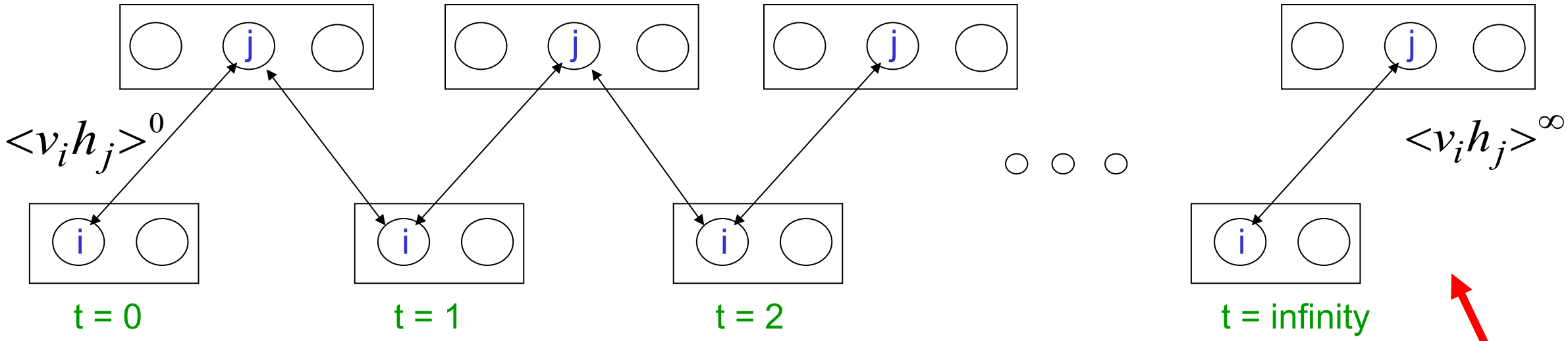
- Die Wahrscheinlichkeit einer Konfiguration von sichtbaren Einheiten

  → ist die Summe der Wahrscheinlichkeiten aller gemeinsamen Konfigurationen die diese enthält.

$$p(v) = \frac{\sum_{h} e^{-E(v,h)}}{\sum_{u,g} e^{-E(u,g)}}$$

→ soll maximiert werden = Lernen

Idee des maximum likelihood Lernalgorithmus für RBM

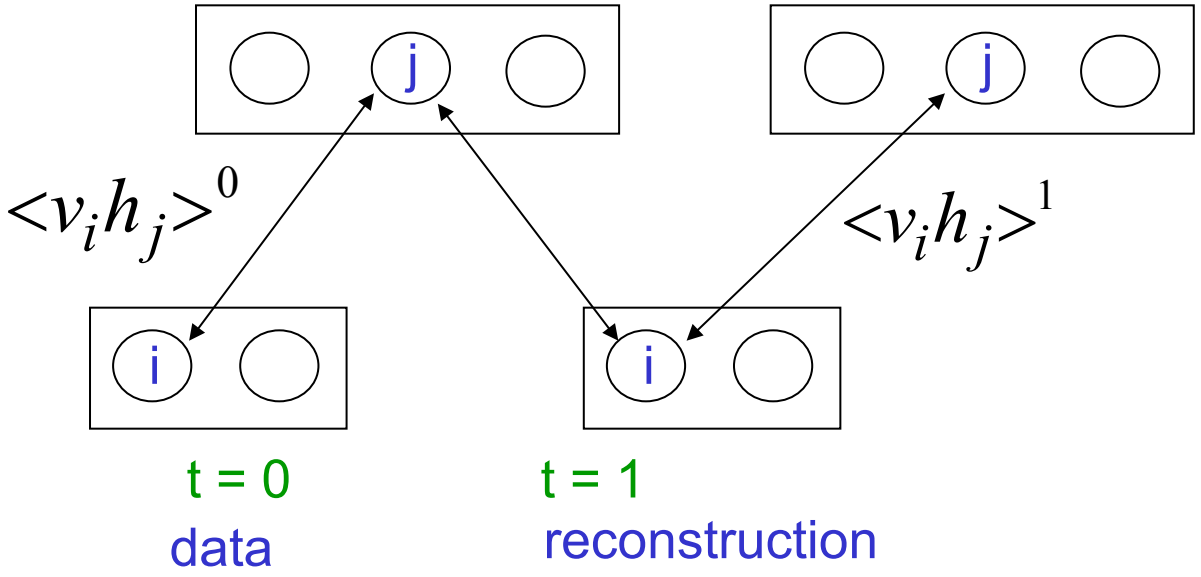


- Start mit einem Trainingsvektor an den sichtbaren Einheiten
- Alternieren zwischen update aller hidden und update aller sichtbaren Einheiten (parallel). (Implementiert „alternating Gibbs“ ← MCMC Ansatz um ein sample der Daten zu erhalten)
- Dann gilt (Gradient):

$$\Delta w_{ij} \approx \frac{\partial \log p(v)}{\partial w_{ij}} = <v_i h_j>^0 - <v_i h_j>^\infty$$

Nicht realistisch

Schneller Lernalgorithmus für RBM (Contrastive divergence learning)



- Start Trainingsvektor an den sichtbaren Einheiten
- Update aller hidden Einheiten - parallel
- Update aller sichtbaren Einheiten (parallel) um eine „Rekonstruktion“ zu erhalten.
- Update der hidden Einheiten

- Update der Gewichte (nach update der hidden Einheiten)

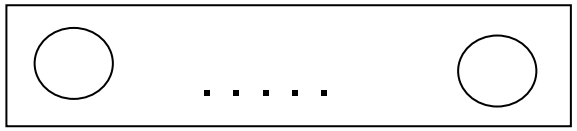
$$\Delta w_{ij} = \varepsilon \left(<v_i h_j>^0 - <v_i h_j>^1 \right)$$

- Entspricht nicht ganz dem Gradienten des log likelihood Aber funktioniert gut.

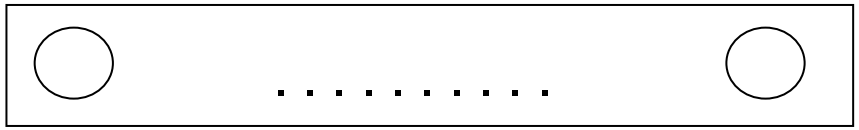
[Carreira-Perpinan & Hinton, 2005]

Beispiel: Abbilden von Ziffern

50 verdeckte Einheiten (feature neurons)

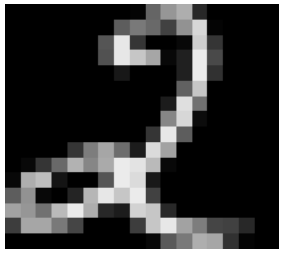


Gewichte W: 50 x (16x16)

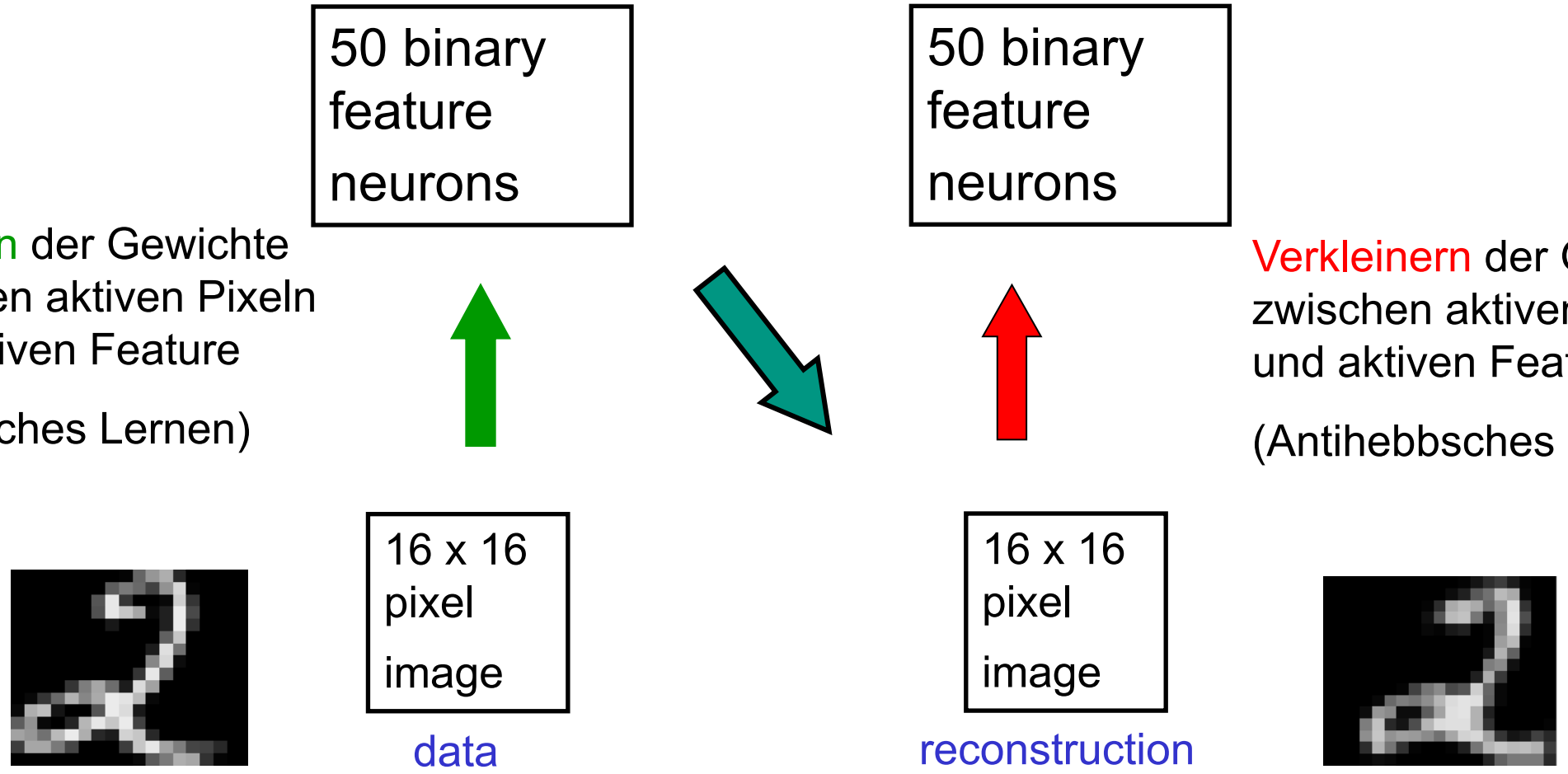


16x16 Einheiten (Pixel)

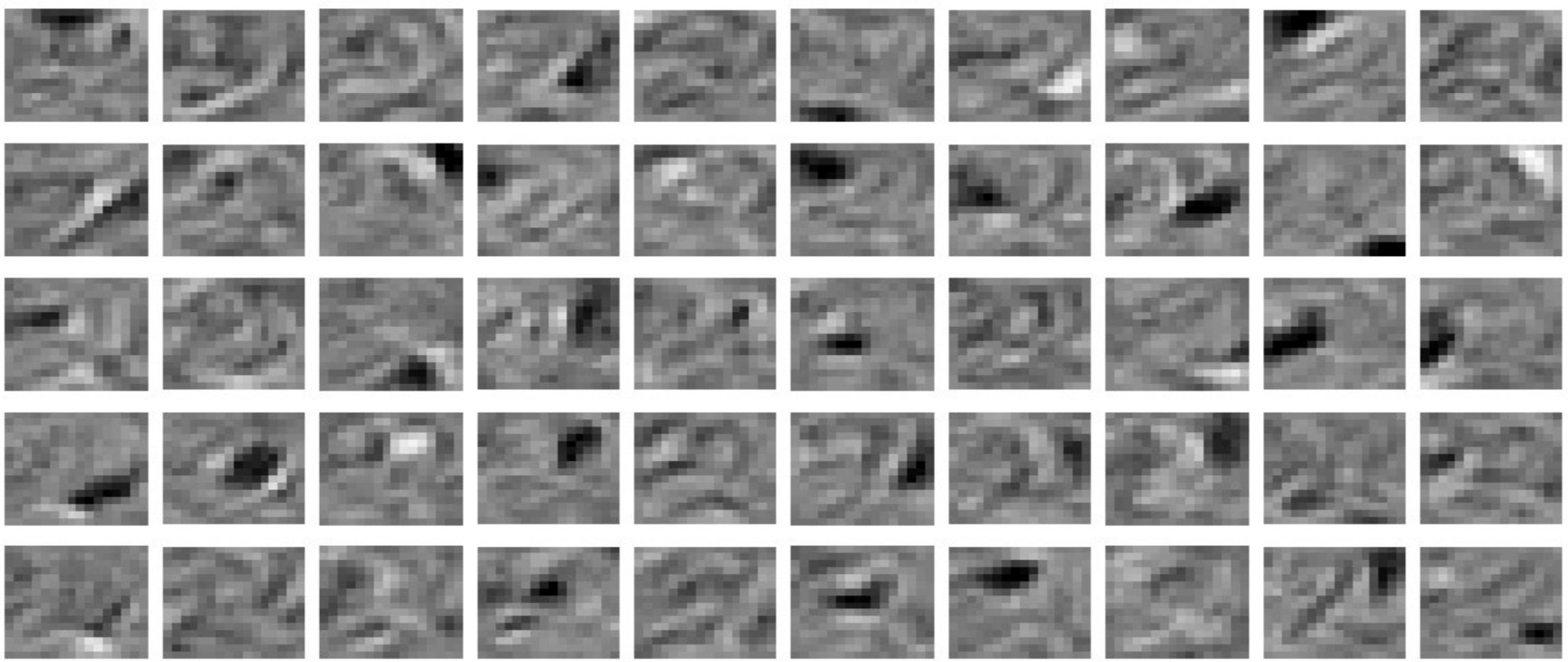
– entsprechend eines Grauwertbildes



Lernen guter “Gewichte” für Ziffer „2“



Die 50 finalen 256 Gewichte (16x16) (Wiederholtes Training)

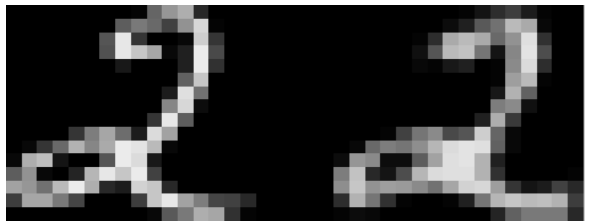


Jede Einheit bildet unterschiedliche Merkmale ab

Wie gut können die Ziffern von binären Merkmalen rekonstruiert werden?

Daten

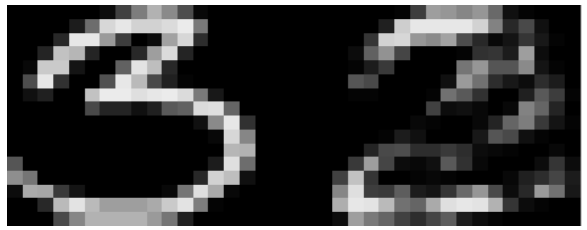
Rekonstruktion mit aktivierten binären Merkmalen



Nächstes Testbild der Ziffernklasse mit dem das Modell gelernt wurde

Daten

Rekonstruktion mit aktivierten binären Merkmalen



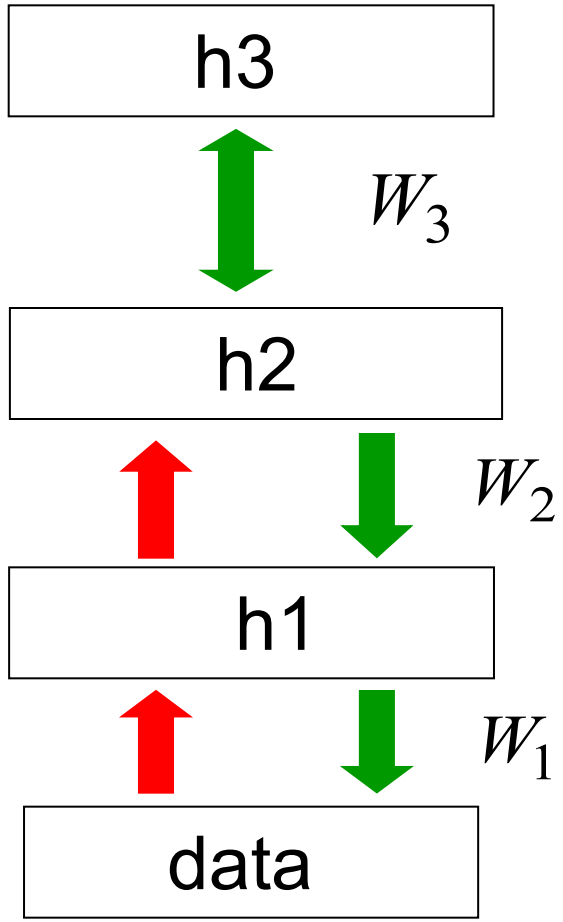
Bilder von unbekannten Daten (d.h. Klassen) → das Netz versucht jedes Bild als 2 zu sehen

Übergang vom RBM zum Tiefen Netz (Deep Net)
(“the main reason RBM’s are interesting” Hinton)

Grundidee: mehrere RBM stapeln

- Tainieren des layer, welcher die Eingaben direkt von den Pixeln erhält (W_1)
- Dann Aktivierung der trainierten Merkmale so verwenden als wären es Pixel und Lernen der zweiten verdeckten Schicht (W_2) usw...
- Es kann gezeigt werden dass jedes Mal wenn ein weitere Schicht hinzukommt die untere Schwelle des log-likelihood der trainierten Daten erhöht wird.
  - Beweis ist nicht trivial

➔ Äquivalenz zwischen RBM und gerichteten Deep Netzen



Wieso funktioniert greedy Lernen

- Die Gewichte W, in der unteren Ebene des RBM definieren $p(v|h)$
- Es gilt für die (untere) RBM

$$p(v) = \sum_h p(h)\ p(v \mid h)$$

- Verbesserung von $p(h)$ wird $p(v)$ verbessern
- Um $p(h)$ zu verbessern benötigen wir ein besseres Modell um die a-posteriori Wahrscheinlichkeit der hidden Einheiten zu generieren ← d.h. die oberen RBM lernen

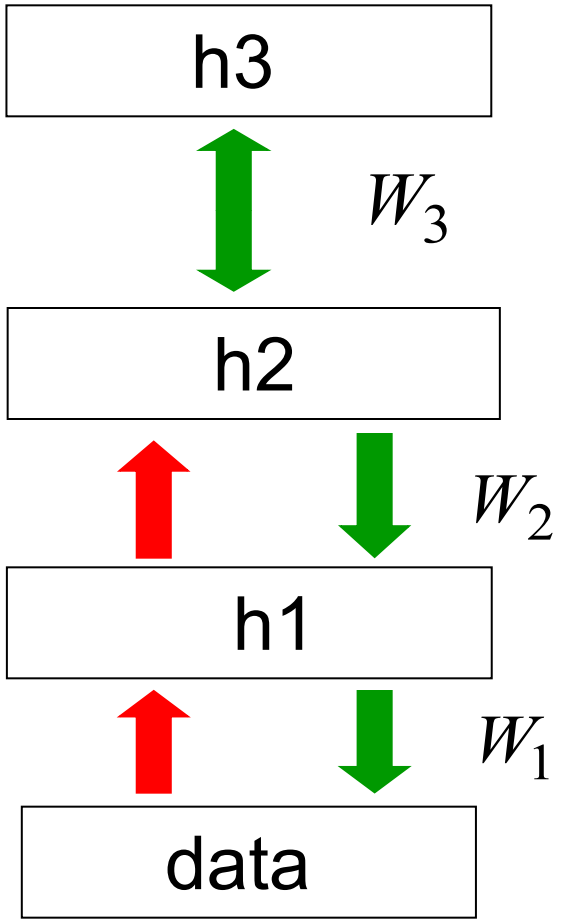
Fine-tuning der Lernens: contrastive “wake-sleep” Algorithmus

- Nach Lernen der einzelnen Layer → Verbessern durch fine-tuning
- Lösen der symmetrischen Bindung zwischen den auf- und abwärts-Gewichten!
- Algorithmus
  1. Wake: Aufwärtsschritt
     - Starte Anregung mit sichtbaren Daten
     - Anpassen der top-down Gewichte um die Aktivität der Merkmale in tiefere Ebenen zu rekonstruieren.
  2. Einige Iterationen in der höchsten RBM
     → Anpassen der Gewichte in der obersten RBM.
  3. Sleep: Abwärtsschritt
     - Anpassen der bottom-up Gewichte um die Aktivität der Merkmale in die höheren Ebene zu rekonstruieren.

Funktionsweise Kaskadiertes RBM - Generativ

- Gegeben: Z.B. ein (eingelerntes) Modell mit 3 Schichten
- Daten generieren:
  1. Anlegen Daten obere Schicht
  2. Einstellen eines Gleichgewichts der obersten RBM durch alternierendes „Gibbs sampling“ (abwechselnd je ein Knoten aktualisieren, über mehrere Iterationen).
  3. Ein Abwärtsschritt um Zustand der anderen Ebenen zu erhalten.

  ➔ Fertig

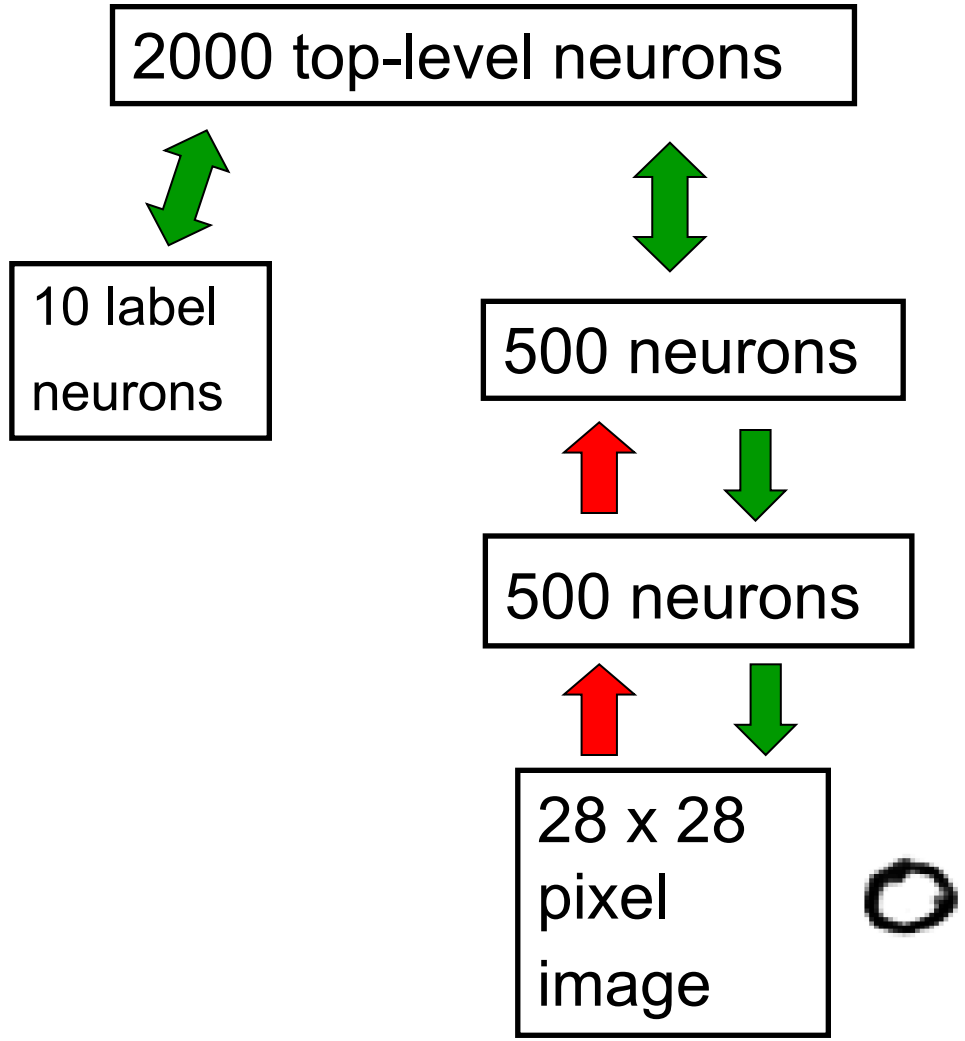


- Die Verbindungen der unteren Schicht nach oben sind nicht Teil des generativen Modells sondern werden für die Inferenz verwendet.

Beispiel: Ziffernerkennung

Die oberste(n) Ebene(n) der RBM
bildet die relativ niedrig-dimensionale Mannigfaltigkeit der Daten ab.

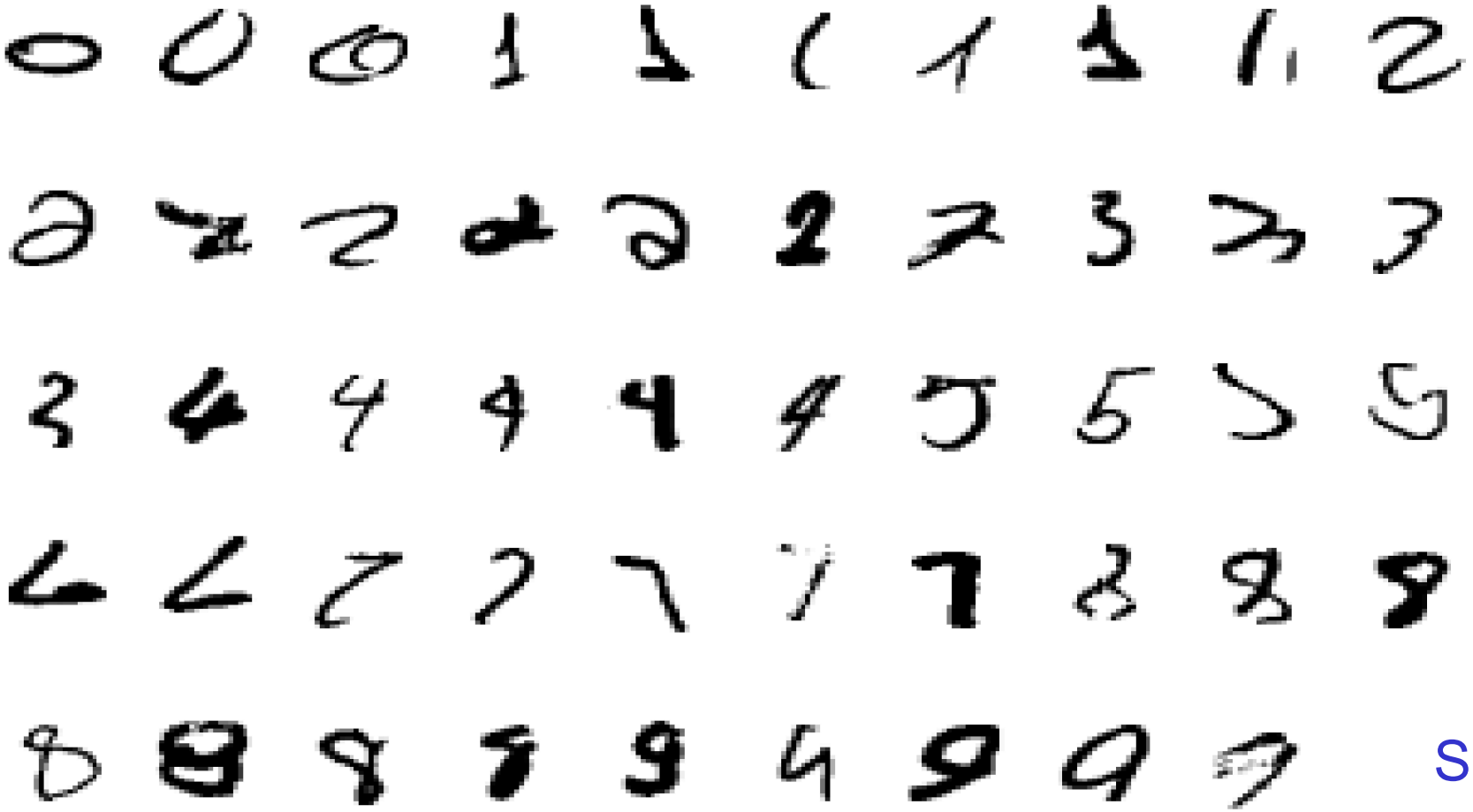
Diese Einheiten sind die „Klassen“



- Das Modell lernt Kombinationen von Label und Bild zu erzeugen.
- Erkennung (Klassifikation):
  - Start mit einem neutralen Zustand der Label Neuronen
  - Aufwärtsschritt von den Bildern hoch
  - Einigen Iterationen in den oberen Ebenen
  - Bestimmen (softmax) aktive Klasse

Beispiel: Ziffernerkennung

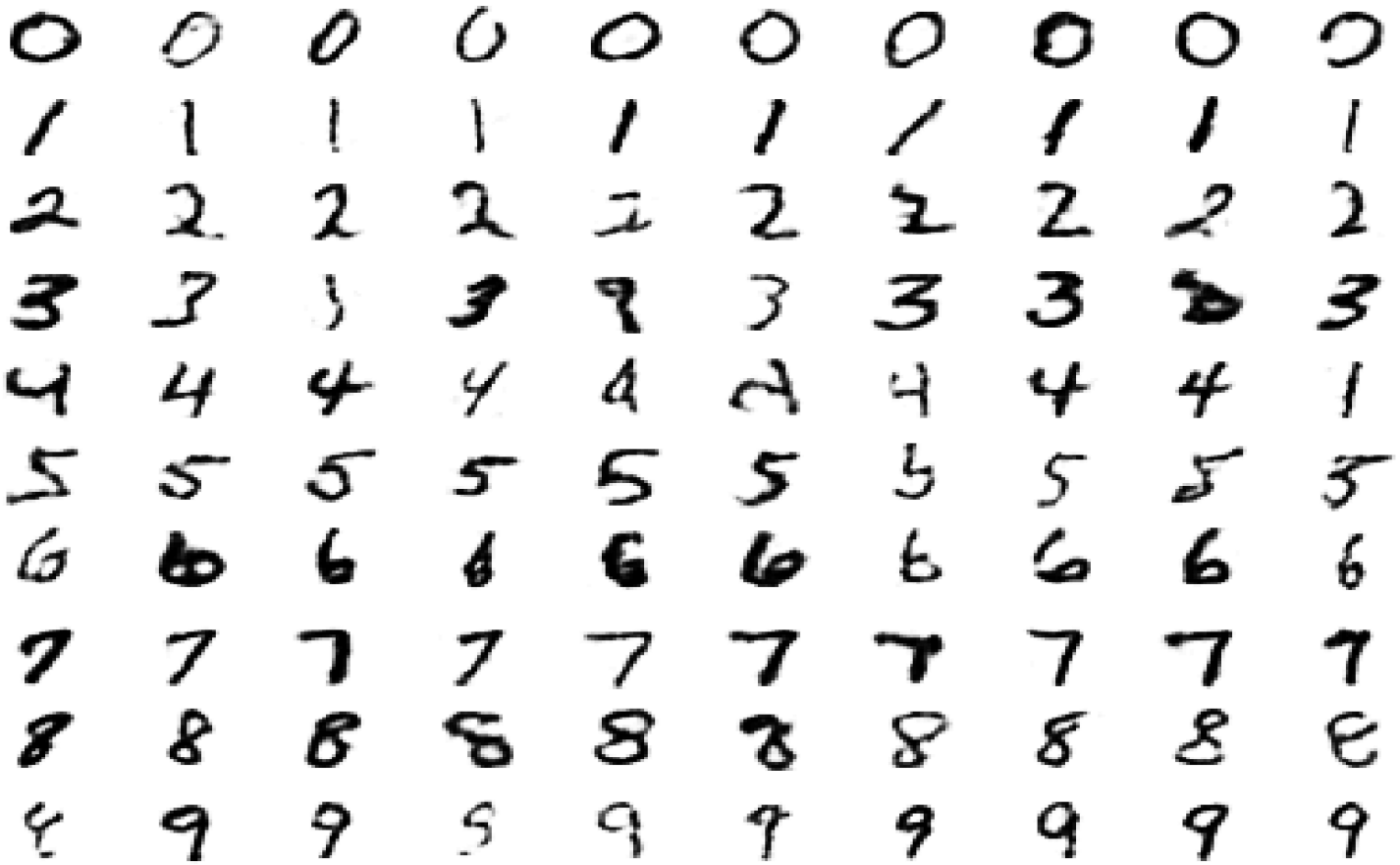
Korrekt erkannte Ziffern die vorher nicht gesehen wurden



Sehr gut !!

Generierte Samples

Jeweilige Label – Einheit fest .
Jeweils 1000 Iterationen ”alternating Gibbs” sampling zwischendurch.

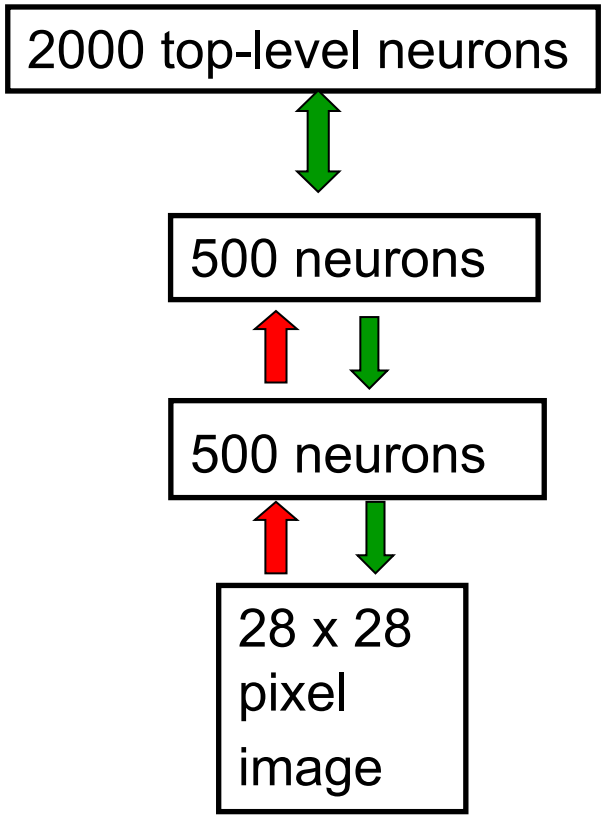


Videos von Netzen die Ziffern generieren

➔http://www.cs.toronto.edu/~hinton/adi/index.htm

Erweiterung: Entscheidungsmodelle Backpropagation? Wie?

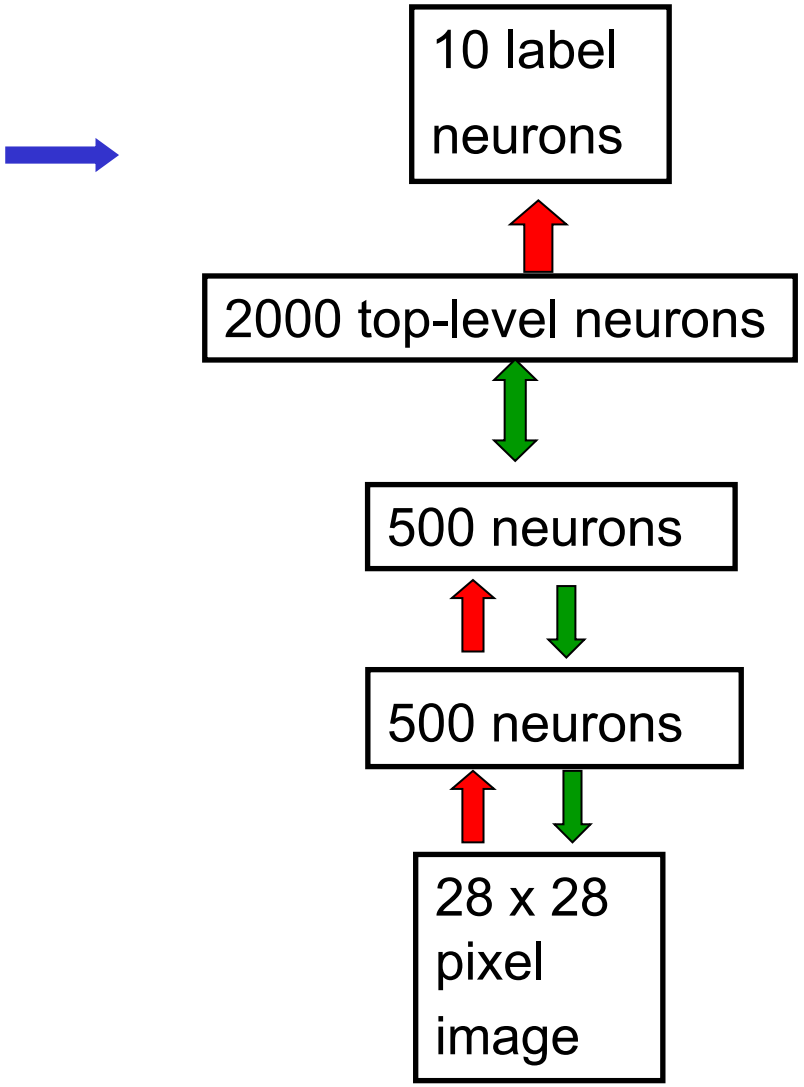
Aber können diese Merkmale auch die Diskrimination unterstützten?



Das Netzwerk lernt zunächst die Verteilung der ungelabelten Daten (Bilder). Wenn man Daten aus dem Modell generiert erhält man gute Ziffern.

Backpropagation? Wie?

Wende für die 10
Einheiten (als oberste Ebene)
backpropagation
(überwacht) an.



Das Netzwerk lernt zunächst die
Verteilung der ungelabelten
Daten (Bilder). Wenn man Daten
aus dem Modell generiert erhält man gute Ziffern.

Fine-tuning für Diskrimination

- Vorgehensweise
  - unüberwacht
    - Lernen je eines layers – „greedily“.
    - Fine-tuning durch contrastive wake-sleep (auf dem vor-trainierten Netz)
  - überwacht
    - Verwenden von Backpropagation
      - Zusätzliche Ebene der Entscheidung (oben)
      - Daten anlegen → Gleichgewicht
      - Label → Backprop → Lernen oberer Gewichte
- Netz ➔ erreicht eine bessere Diskriminationsfähigkeit
- Besser als Standard-Backpropagation auf NN

Warum ist backpropagation besser nach gready pre-training

- Gready Lernen funktioniert gut auch für große Netze.
- Backpropagation startet erst nachdem gute Gewichte für die Daten gefunden wurden.
  - Die Gradienten oberer Schichten sind “sensibel” und backprop wird nur eine lokale Suche durchführen.
  - Viel Information in den finalen Gewichten kommt von der guten Modellierung der Datenverteilung der Input Vektoren.
  - Die Information der Labels ändert die unteren Gewichte nicht.
- Diese Methode des backpropagation funktioniert gut auch wenn die Mehrzahl der Daten ungelabelt ist, weil ungelabelte Daten die guten Merkmale bereits bereitstellen.

Validierungsbeispiel: Fehler auf dem MNIST Testdatensatz

- Generative model of unlabelled digits followed by gentle backpropagation (Hinton & Salakhutdinov, Science 2006) 1.15%
- Generative model based on RBM's 1.25%
- Support Vector Machine (Decoste et. al.) 1.4%
- Backprop with 1000 hiddens (Platt) ~1.6%
- Backprop with 500 →300 hiddens ~1.6%
- K-Nearest Neighbor ~ 3.3%
- [Le Cun et. al. 1998] – weitere Ergebnisse

Fazit (so weit)

- RBM liefern eine einfache Methode um mehrere Ebenen von Merkmalen unüberwacht zu lernen.
  - Echtes Maximum Likelihood Lernen ist aufwendig (nicht möglich)
  - Aber s.g. Contrastive Divergence Lernen ist schnell und gut
- Mehrere Ebenen können gut gelernt werden
  - wenn die verdeckten Zustände unterer RBM als sichtbare Einheiten der nächsten RBM betrachtet werden
- Dies führt zu guten generativen Modellen die noch verfeinert werden können (fine-tune).
  - Contrastive wake-sleep.
  - Diskriminatives fine-tuning

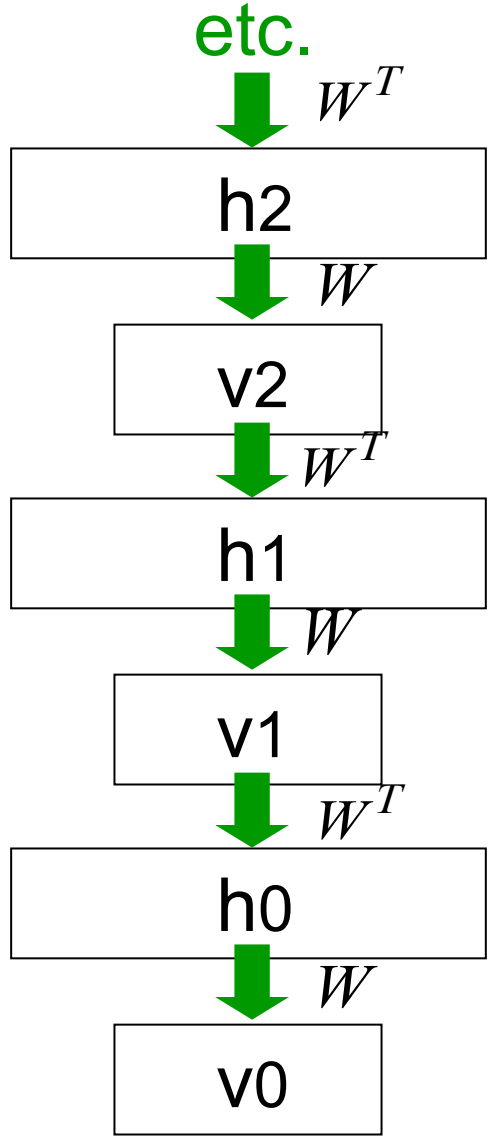
Übersicht

- Motivation
  - Belief Netze
  - ➔ Nutzen
  - ➔ Prinzip
  - Problematik des Lernens
- Restricted Boltzmann Maschinen (RBM)
- Tiefe Netze mit RBM (DBM)
- Äquivalenz zu Sigmoiden Belief Netzen — s. Folien
- Anwendungsmöglichkeiten

Sigmoid Deep Belief Netze = Kaskadiertes RBM = Deep RBM?

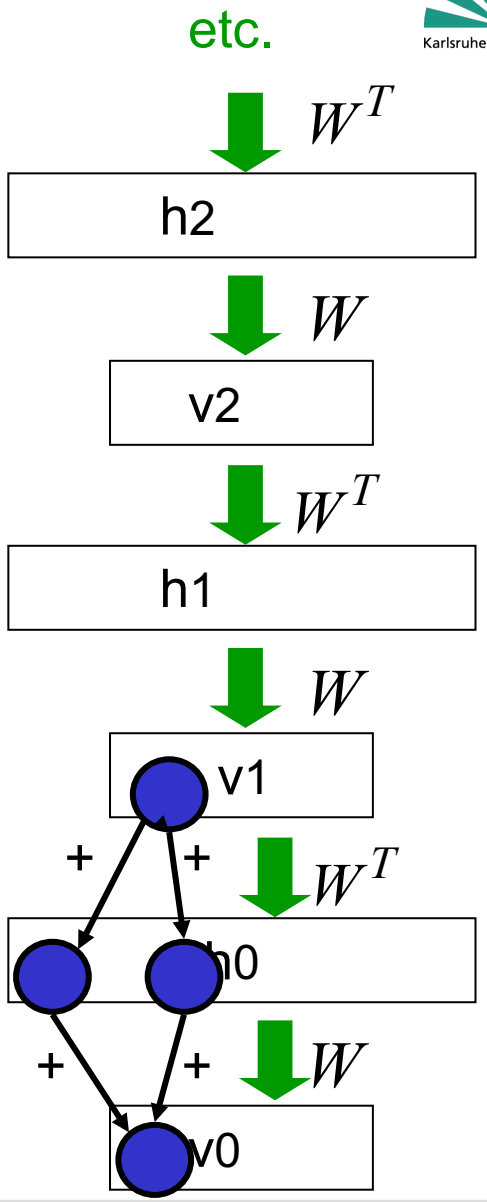
- Es gibt unendliche Sigmoid Belief Netze die äquivalent zu einem RBM sind

- Gedankenexperiment ➔ Zeitliches Aufrollen eines 2-schichtigen RBM in ein unendlichen gerichteten Graphen
- W – gekoppelt , Dimensionen der v und h Ebenen gleich
- die bedingten Wahrscheinlichkeiten $p(v|h)$ und $p(h|v)$ sind definiert durch W
- Ein Abwärtsschritt im gerichteten Netz ist genau äquivalent dazu das RBM Netz ins Gleichgewicht zu bringen.



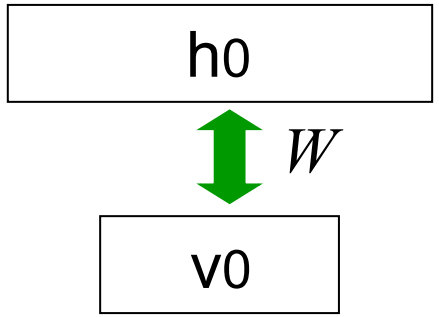
Inferenz in einem gerichteten Netz mit wiederholten Gewichten

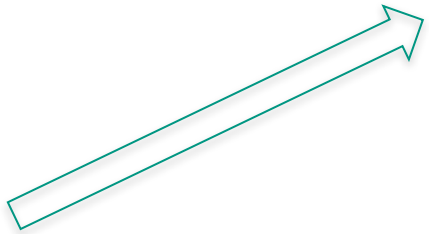
- Inferenz: Multipliziere v_0 mit W^T+ sigmoid Aktivierung
- Das Modell oberhalb h_0 implementiert zusätzliche a-priori Daten
- Inferenz in einem gerichteten Netz ist äquivalent dem Fall eine RBM ins Gleichgewicht zu bringen beginnend mit sichtbaren Daten.

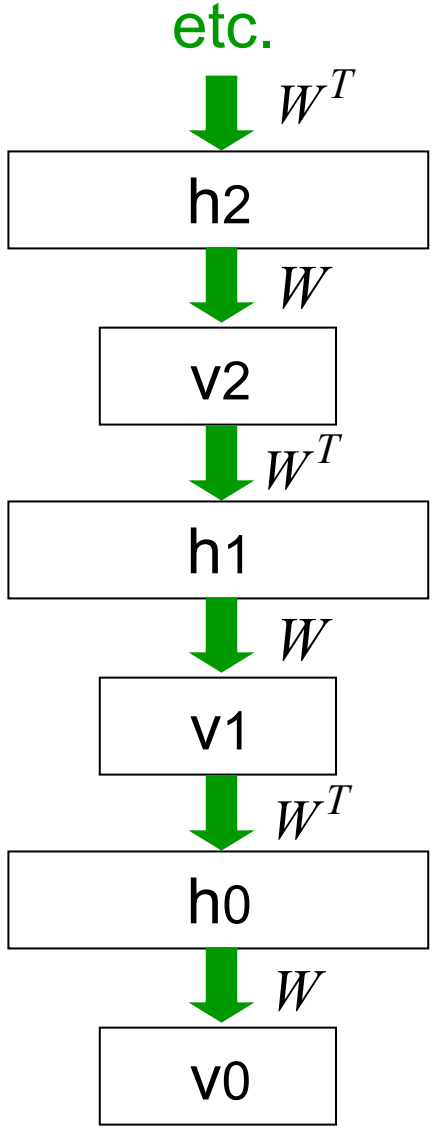


Lernen in einem gerichteten Deep Netz

- Zunächst Lernen mit gekoppelten Gewichten
  - Entspricht Lernen von RBM
  - Ignorieren kleiner Abweichungen durch die gekoppelten Gewichte der höheren Ebenen

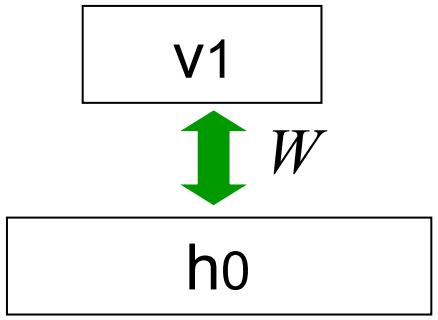


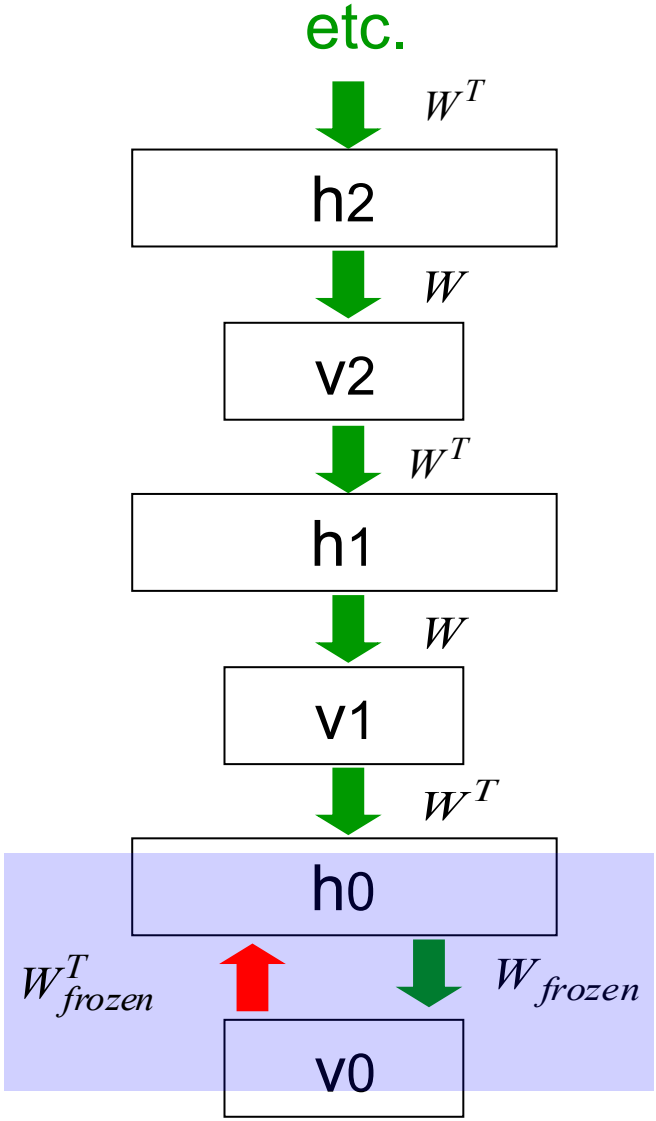




Lernen in einem gerichteten Deep Netz

- Anschließend Einfrieren des ersten Layer in beide Richtungen und Lernen der nächsten Gewichte (diese sind immer noch gekoppelt).
  - Äquivalent dem Lernen einer weiteren RBM unter Verwendung der aggregierten a-posteriori Verteilung von h_0 als Daten.





Ist dies korrekt?

- Nicht ganz aber hinreichend
  - Inferenz mit entkoppelten Gewichten der Layer ist nicht wirklich korrekt denn es entspricht nicht mehr einer RBM ...
  - Hinton, Osindero and Teh (2006) zeigten, dass dies eine gute Approximation (Verbesserung) ist

- Fazit: Kaskadierte RBM – gute Annäherung an Deep Belief Netze

Übersicht

- Motivation
  - Belief Netze
  - ➔ Nutzen
  - ➔ Prinzip
  - Problematik des Lernens
- Restricted Boltzmann Maschinen (RBM)
- Tiefe Netze mit RBM (DBM)
- Äquivalenz zu Sigmoiden Belief Netzen
- Anwendungsmöglichkeiten

Wie viele Ebenen, welcher Größe verwenden?

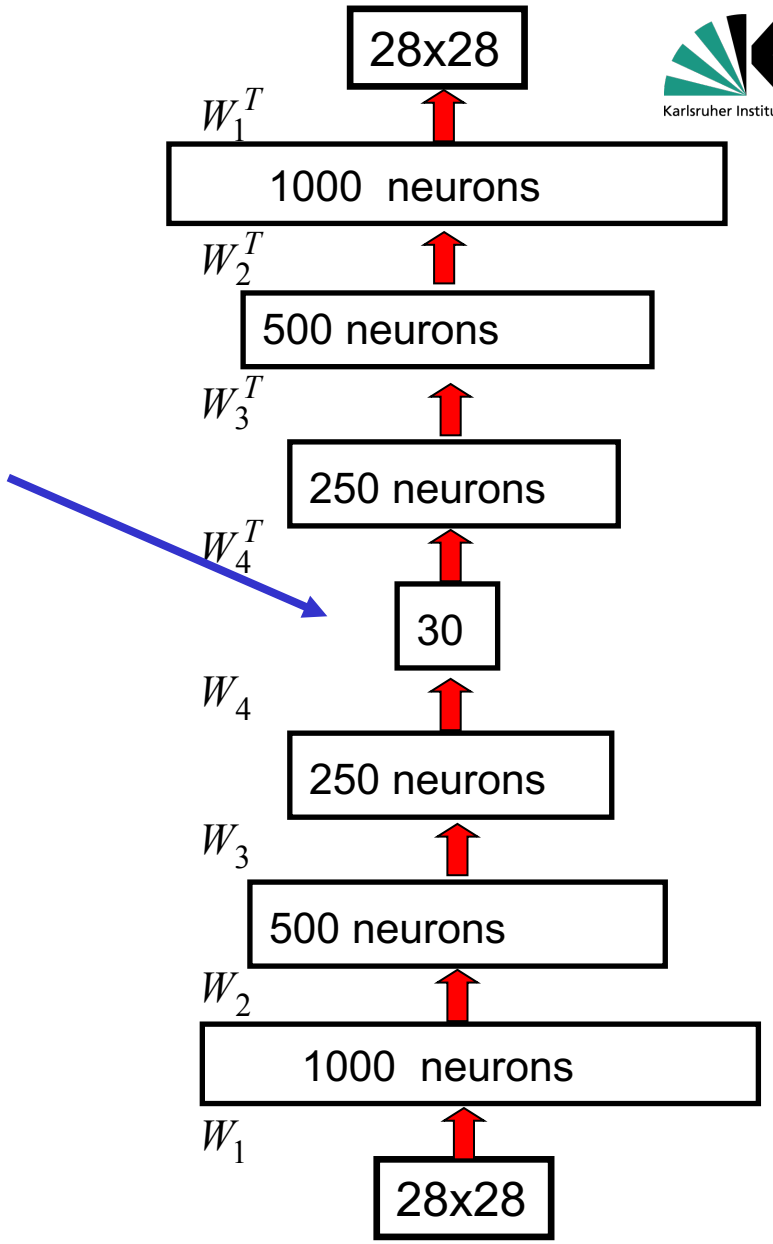
- Deep Belief Netzte lassen dem Entwickler viel Freiheit
  - ➔ Größe abhängig von der Aufgabe ☺ / ☹
  - Mit genügend Ebenen kann jede Verteilung über binäre Daten - Vektoren abgebildet werden (Sutskever & Hinton, 2007)

- Hinton: „If freedom scares you, stick to convex optimization of shallow models that are obviously inadequate for doing Artificial Intelligence.“

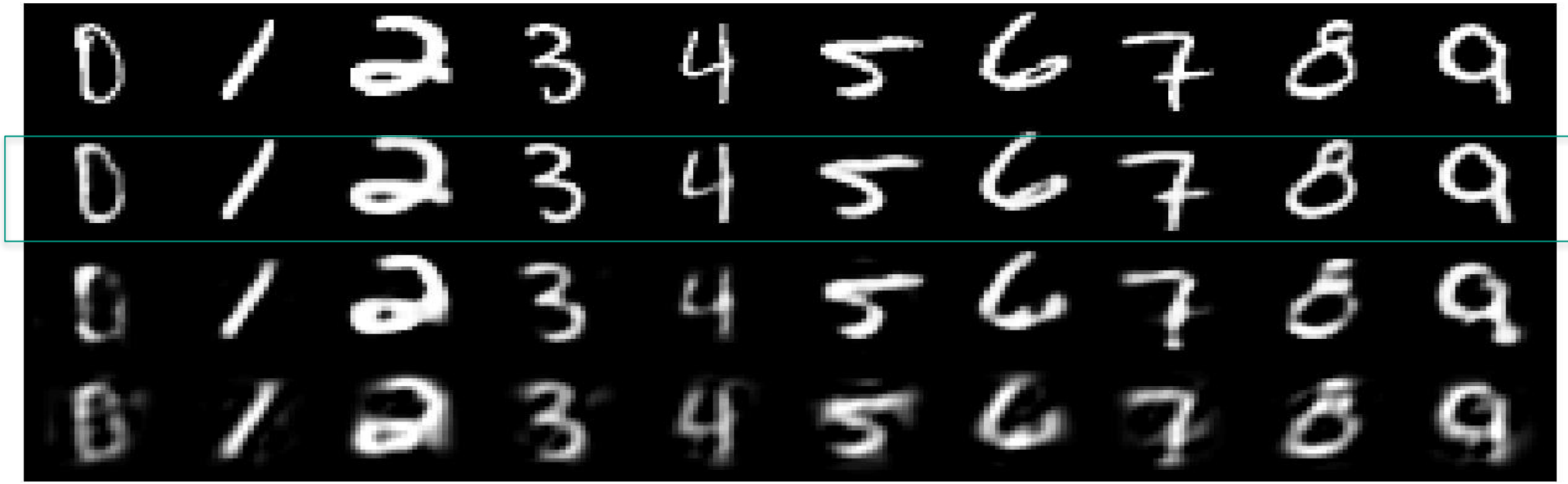
Deep Autoencoders
(Hinton & Salakhutdinov, 2006)

- Nutzen: Methode um nichtlineare Dimensionsreduktion zu erhalten
  - NN + Backpropagation → funktioniert nicht so wirklich

- Bessere Methode :
  - Trainiere stack von 4 RBM's
  - Ausrollen
  - Fine-tune mit backprop.



Vergleich: Komprimieren von Bildern auf 30 Dimensionen.



real data

30-D deep auto

30-D logistic PCA

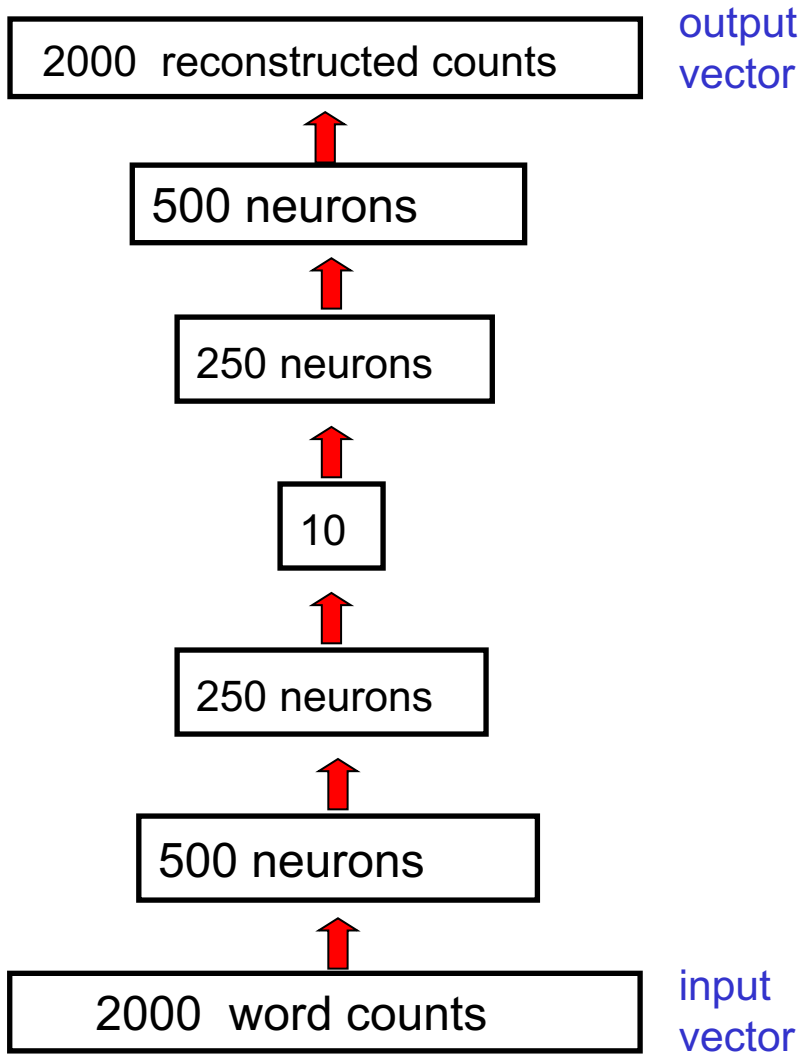
30-D PCA

Anwendung: Ähnliche Dokumente (query document)

- Generierung niedrig dimensionaler Codes für Dokumente
  - Ziel eine schnelle und genaue Identifizierung ähnlicher Dokumente (aus einer großen Menge).

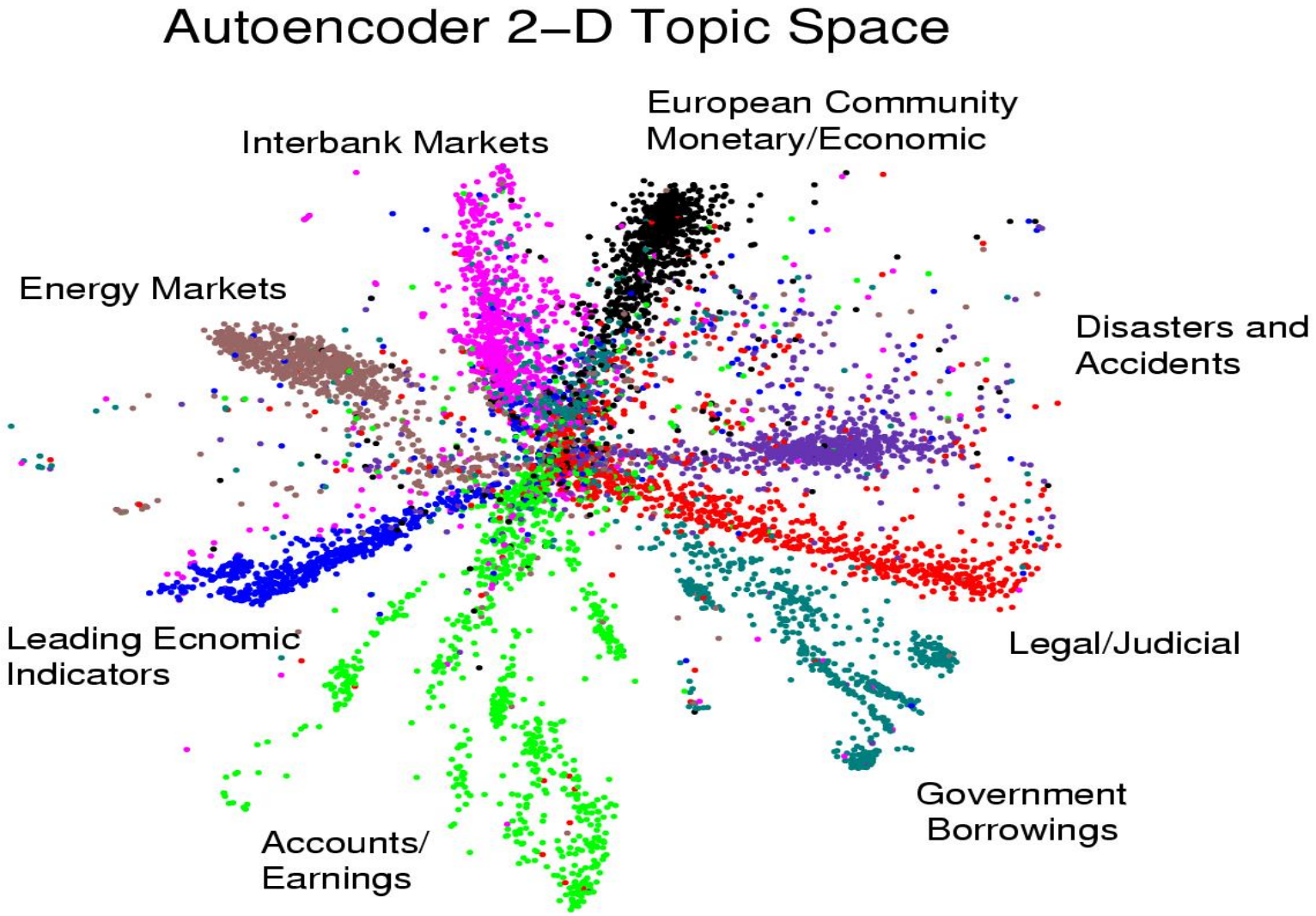
- Idee
  - Konvertierung jedes Dokument in “bag of words”
    - Dies ist ein 2000 dimensionale Vektor der die Anzahl der 2000 geläufigsten Worte beinhaltet
  - Weitere Reduzierung mit Autoencoder

Komprimieren



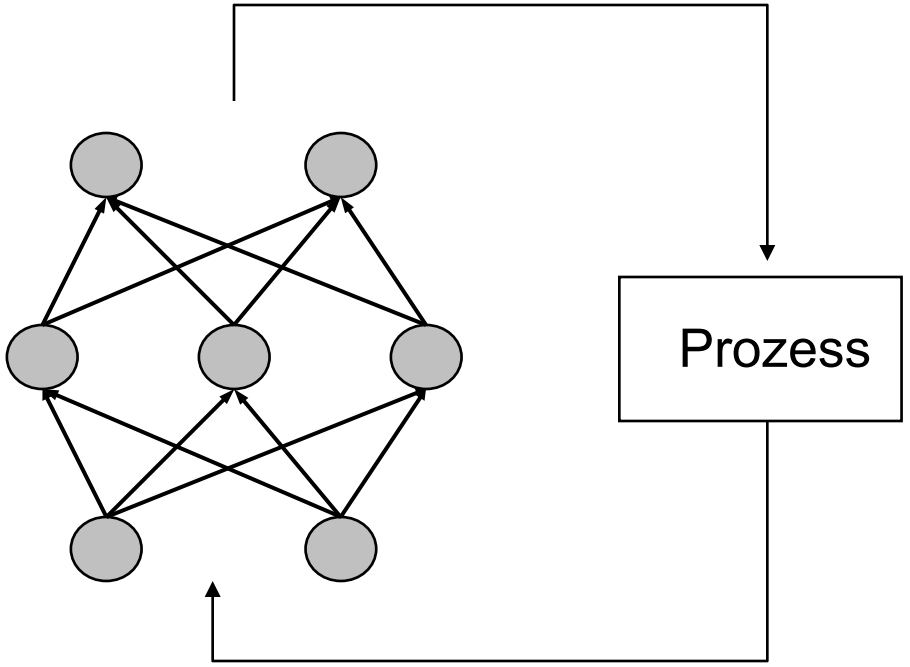
- Trainieren eines Netzes das die Eingabe als Ausgabe reproduziert
- Die Information wird in 10 Einheiten abgebildet
  - auch „Zentraler Botttelneck“ genannt
- Diese bilden dann eine gute Abbildung um Dokumente zu vergleichen

Komprimierung auf 2 D



Repräsentation der Zeit (für Raum ähnlich)

- Problemstellung:
  - zeitlich veränderliche Eingabe
  - Ausgabe hängt von der Änderung der Eingabe (über eine längere Zeit) ab
- Lösungsansatz:
  - Lernen mit Gedächtnis
- Anwendung:
  - Steuerung
  - Generierung oder Klassifikation von Zeitreihen



Möglichkeit: Zeitfenster-Technik

Eingabe:

Eingabewerte der letzten n Zeitpunkte

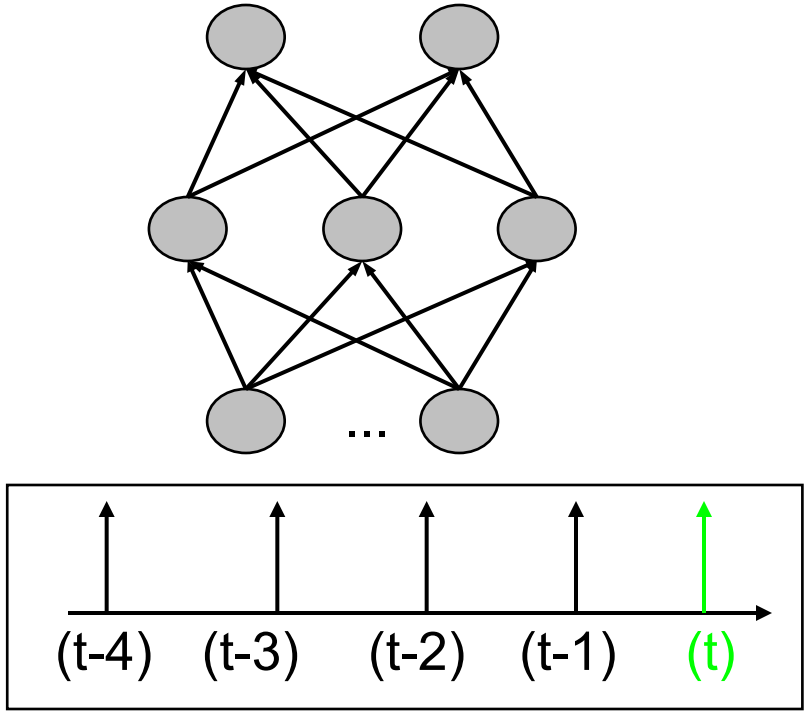
Ausgabe:

für den aktuellen Zeitpunkt

Lernen:

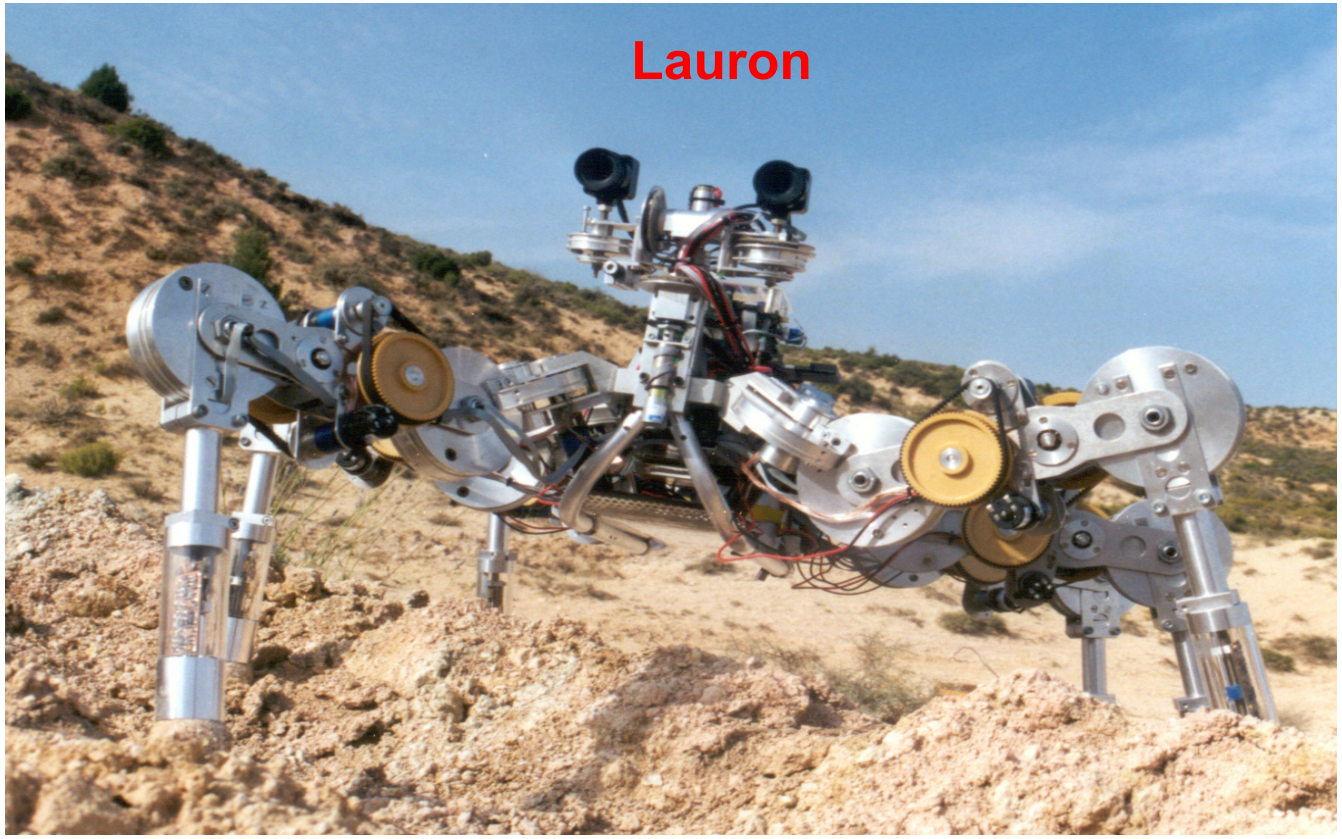
so, als wären es höher-dimensionale Daten

Schon lange verwendet !!
(Bsp. siehe nächste Folien)



Zeitfenster

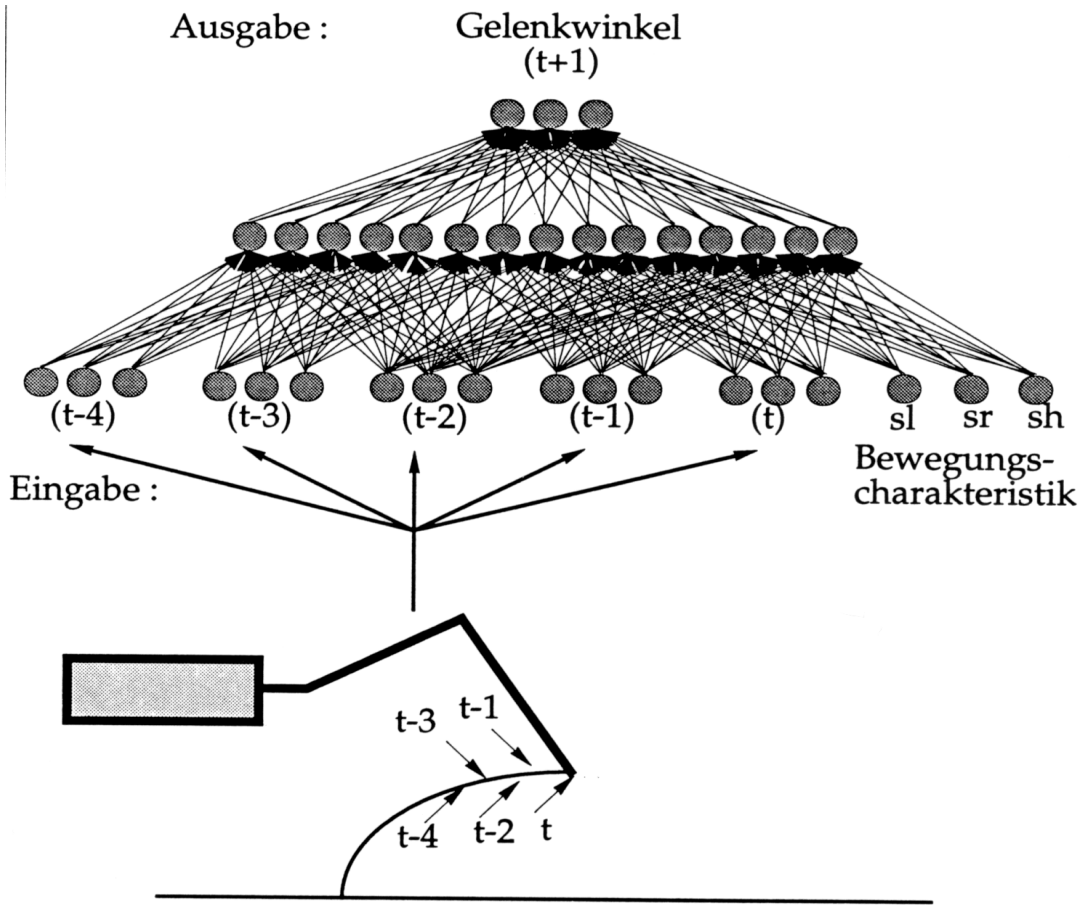
Frühe Anwendung: Lauron mit NN



Anwendung: Bewegungssteuerung

Steuerung der einzelnen Beine (Gelenkwinkel)

Lauron (viel früher): Zeitfenster als Netzeingabe



Einlernen von Beintrajektorien zur Beinsteuerung einer sechsbeinigen Laufmaschine (Ilg et al. ´90-er)

Eingabe Zeitfenster:
- 4 letzten Gelenkwinkel

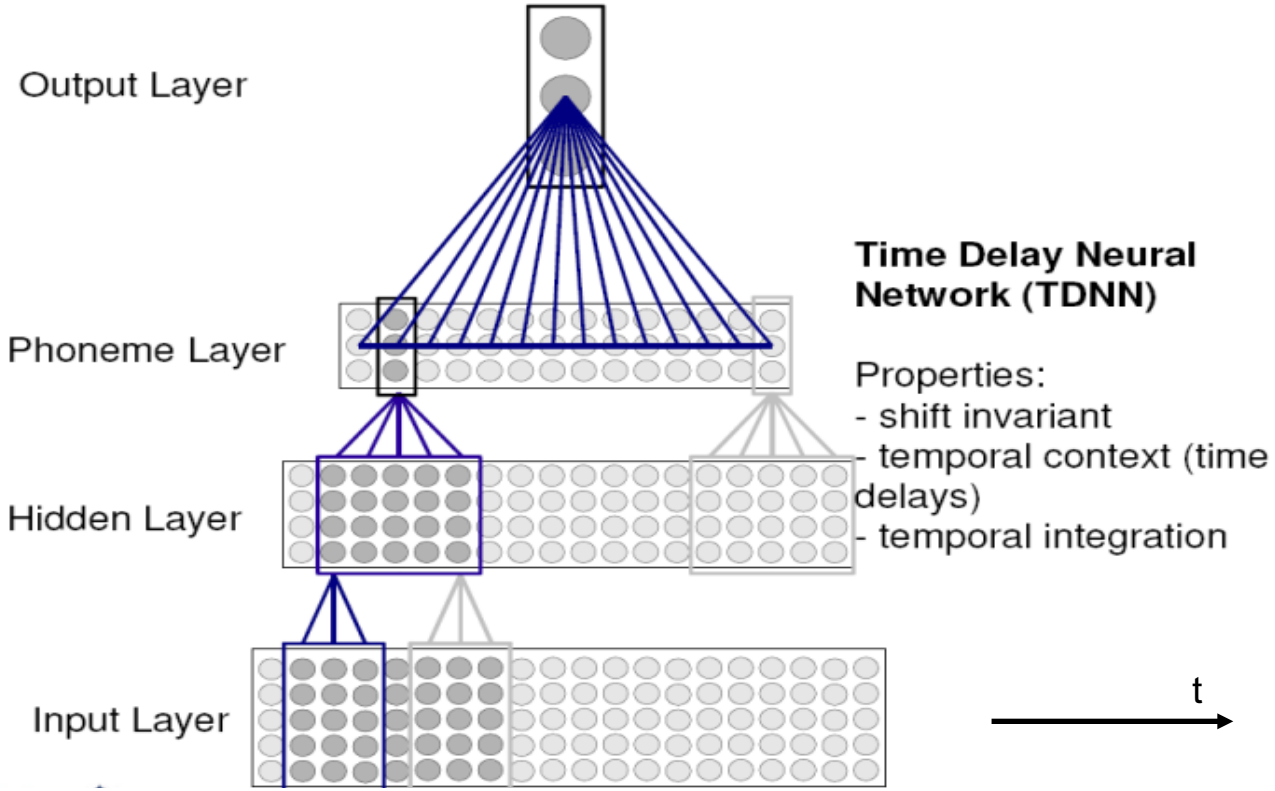
Weitere Eingaben:
- Bewegungsparameter

Ausgabe:
- Gelenkwinkel

Ergebnis:
- grundsätzlich gut

TDNN - Time Delay Neural Networks (Waibel89 - Spracherkennung)

- Erweiterung der Zeitfenstertechnik: Integration mehrere NN - Instanzen im output layer

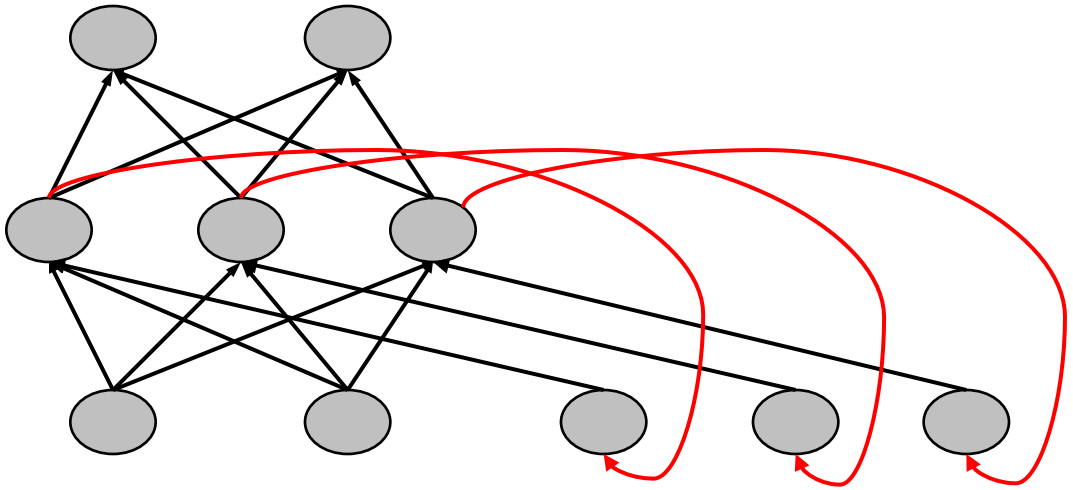


- Wichtigster Aspekt: zeitliche Verschiebungsinvarianz

Frühe Rekurrente Netzarchitekturen (NN)

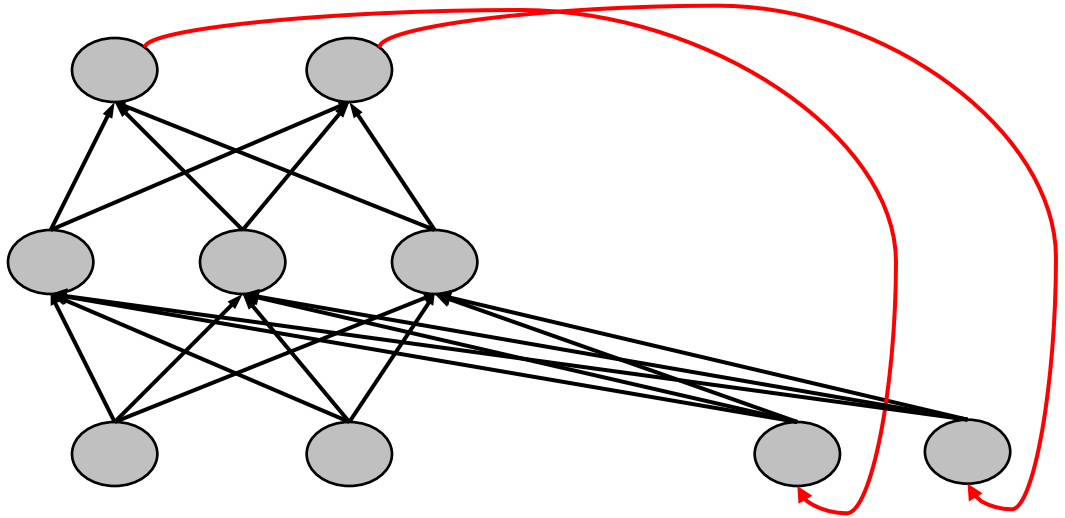
Gedächtnis durch
Rückkopplung der Neuronenausgaben

Rückkopplung der
Hidden - Schichten



Elman Netzwerk (Elman1991)

Rückkopplung der
Ausgabeneuronen

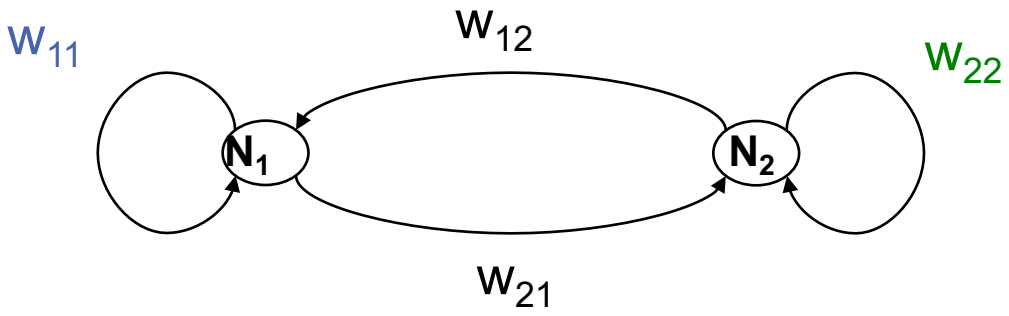


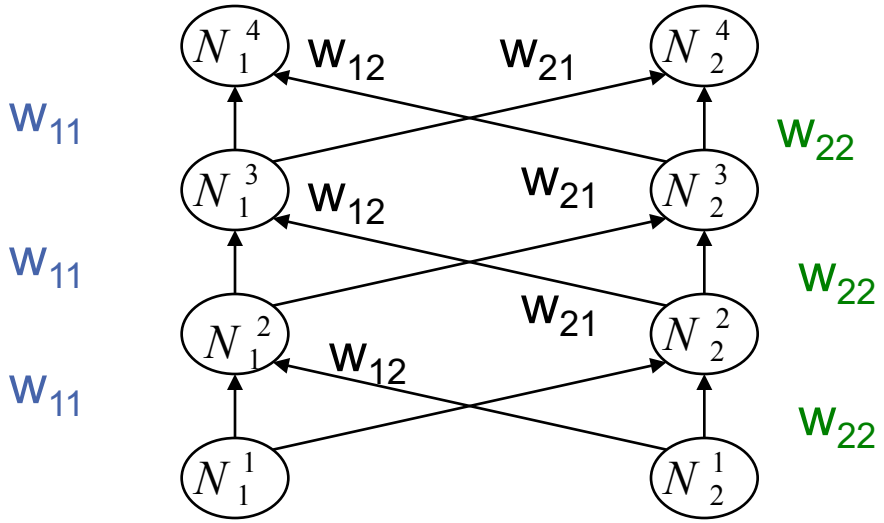
Jordan Netzwerk (Jordan1991)

Lernen bei frühen Rekurrenten NN – Netze

Backpropagation through time

- „Ausrollen“ des rekurrenten Netzes
- Kopplung der Gewichte
- BackProp





$$o_i(t+1) = f(net_i(t)) = f\left(\sum_j w_{ij} o_j + x_i(t)\right)$$

stabil bei begrenzten kurzen Sequenzlängen

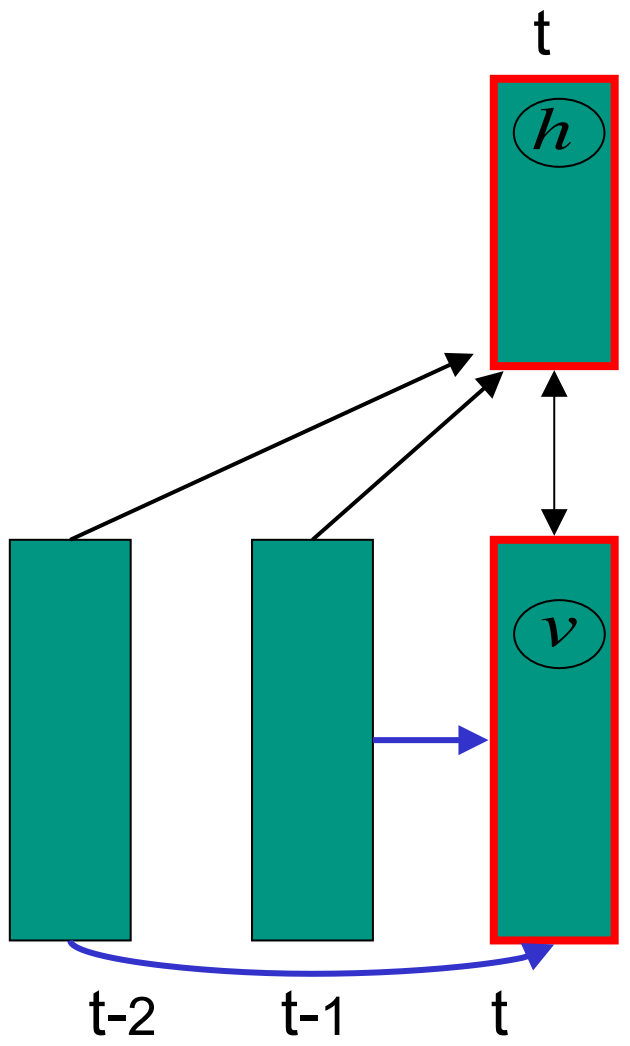
Deep Learning - Zeitreihenmodelle

- 3 Tricks :
  - Verwende RBM
  - Modelliere (kurzzeit-) temporale Information durch „Gedächtnis“ d.h. vorangegangene Zustände
  - Diese bilden einen zusätzlichen Input für die hidden und die sichtbaren Einheiten

- Dies führt zu einem temporalen Modell das gestapelt werden kann
  - Man erhält ein Greedy – Deep - Lernmodell für temporale Strukturen

Conditional RBM Modell (Sutskever & Hinton 2007)

- Inputs der früheren Zustände der sichtbaren Einheiten bilden dynamische Bias für die aktuellen hidden und sichtbaren Einheiten
- Rekonstruieren der Daten zum Zeitpunkt t aus dem Zustand der hidden Einheiten und den früheren Zuständen der sichtbaren Daten
- Analog für hidden Einheiten
- Lernen erfolgt ähnlich wie gehabt durch contrastive divergence
- Alternating Gibbs sampling für einige Iterationen zwischen hidden units und sichtbaren Einheiten erzeugt neue hidden und sichtbare Zustände
  (die kompatibel zueinander und mit der Historie sind)

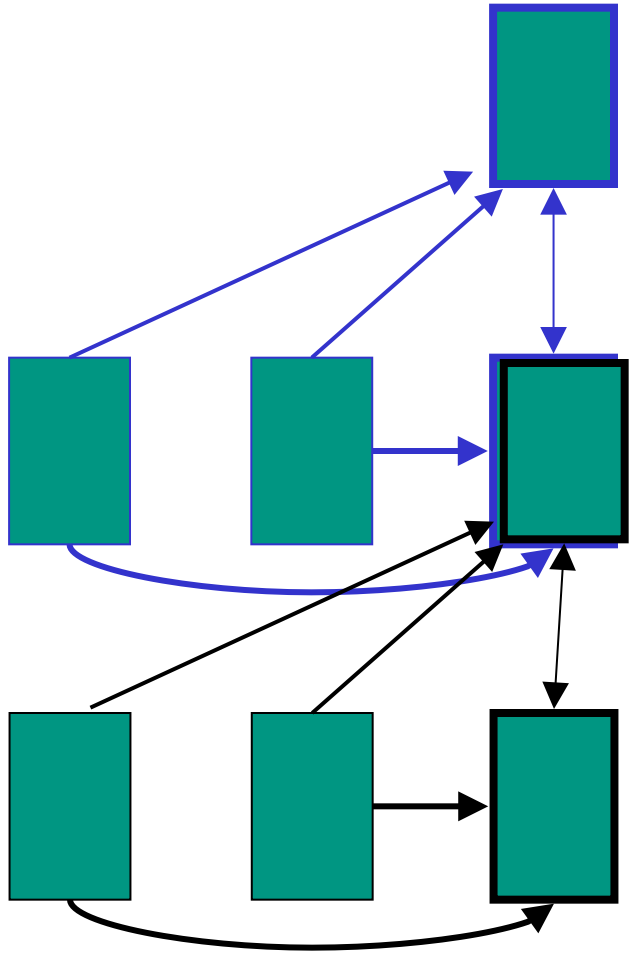


Stapeln von temporalen RBM's

- Wie gehabt: Betrachte die hidden Aktivitäten der ersten Ebene als Daten der zweiten Ebene.

- Nach gready Lernen können Daten aus dem gesamten Modell generiert werden
  - Generiere beginnend beim obersten level (alternating Gibbs sampling) zwischen aktuellem hidden und sichtbaren Einheit der oberen RBM unter Verwendung der dynamischen bias der sichtbaren Einheiten

  - Abwärts-Schritt zu den niedrigen Layern unter Verwendung der s.g. autoregressiven inputs der früheren Einheiten in jedem Layer

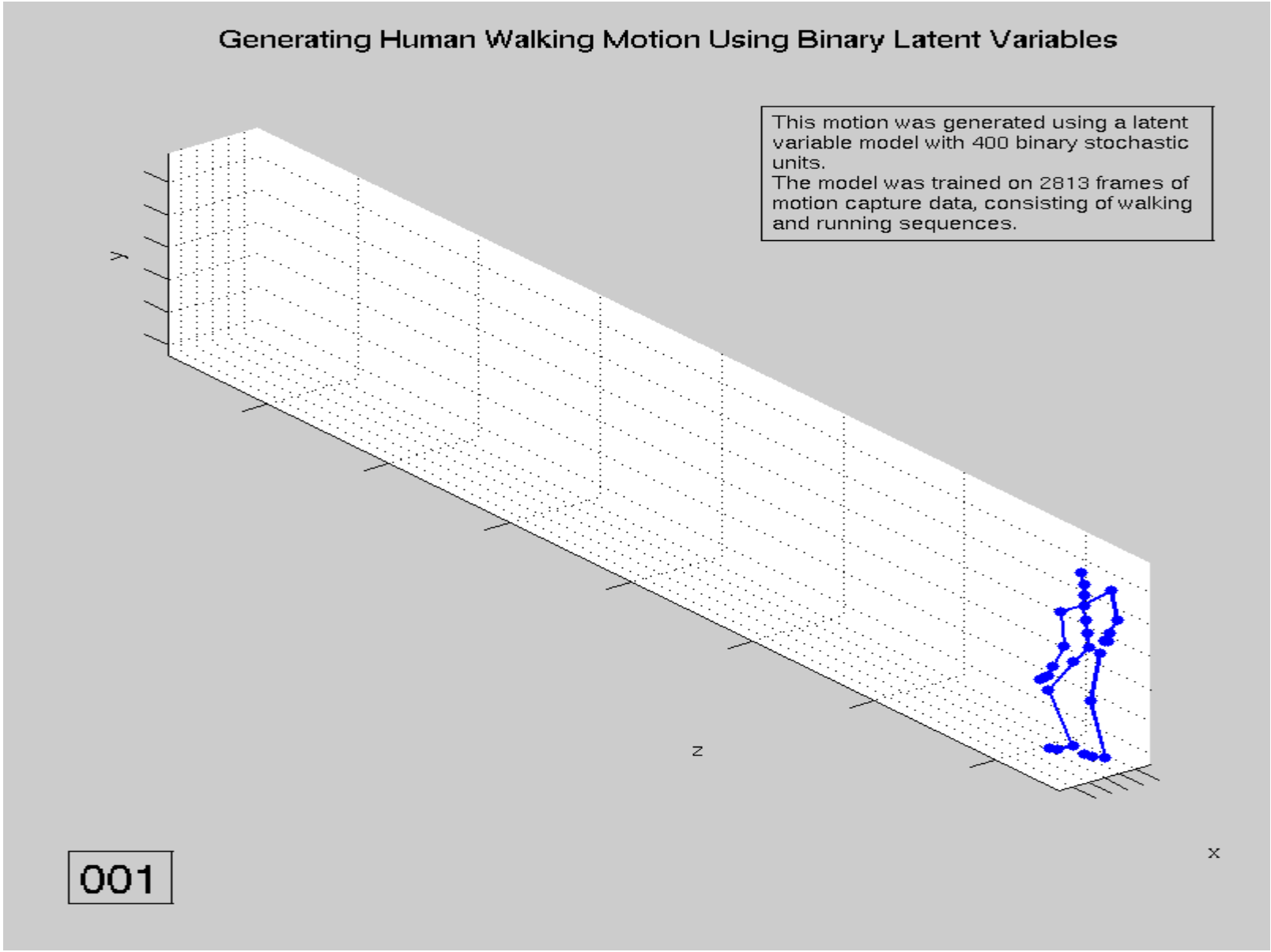


Anwendung: Lernen aus motion capture data (Taylor, Roweis & Hinton, 2007)

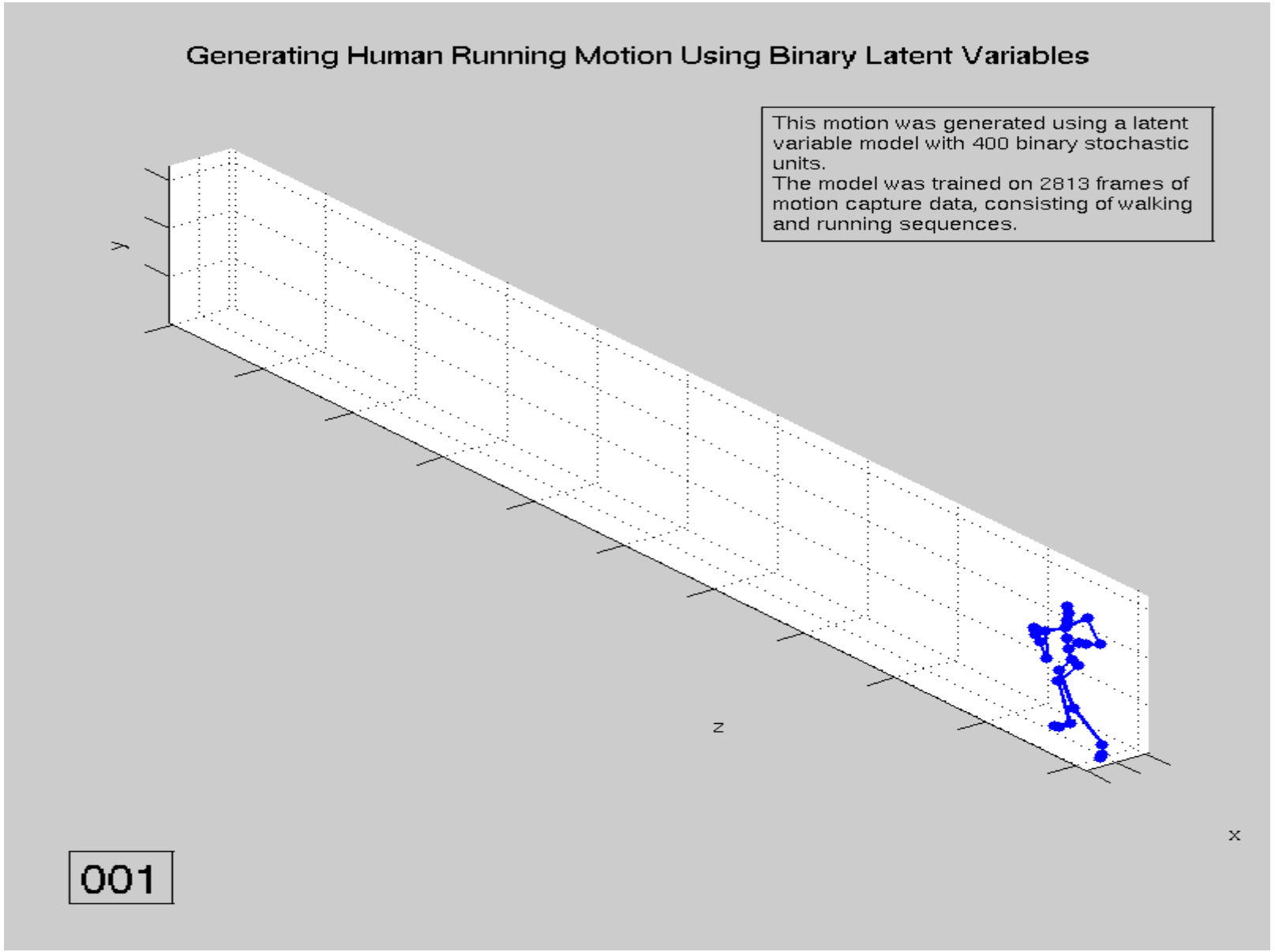
- Menschliche Bewegungen werden über reflektive Marker und infrarot Kameras aufgezeichnet
- Gegeben ein Skelettmodell können die 3D Positionen der Marker in Gelenkbewegungen + Position des Körpers konvertiert

Bild ähnlich

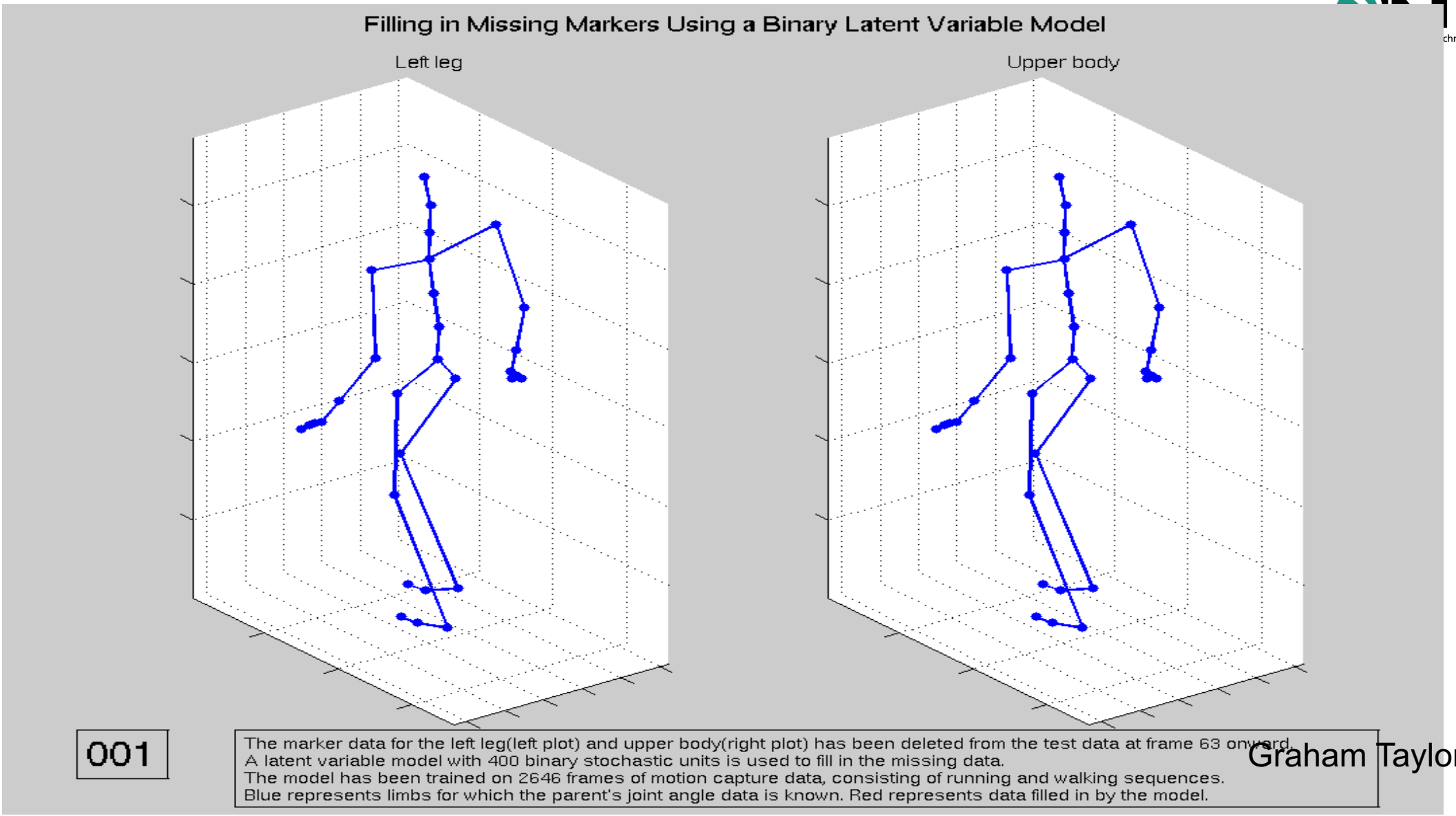
- Ziel: Lernen eines Modells für Gehen, Laufen, ...
  - gemeinsame Wissensrepräsentation aller Bewegungen
  - Transitionen zwischen Gehen und Laufen
  - Online Inferenz →fehlende Marker (Bewegungen) werden kompensiert



Graham Taylor



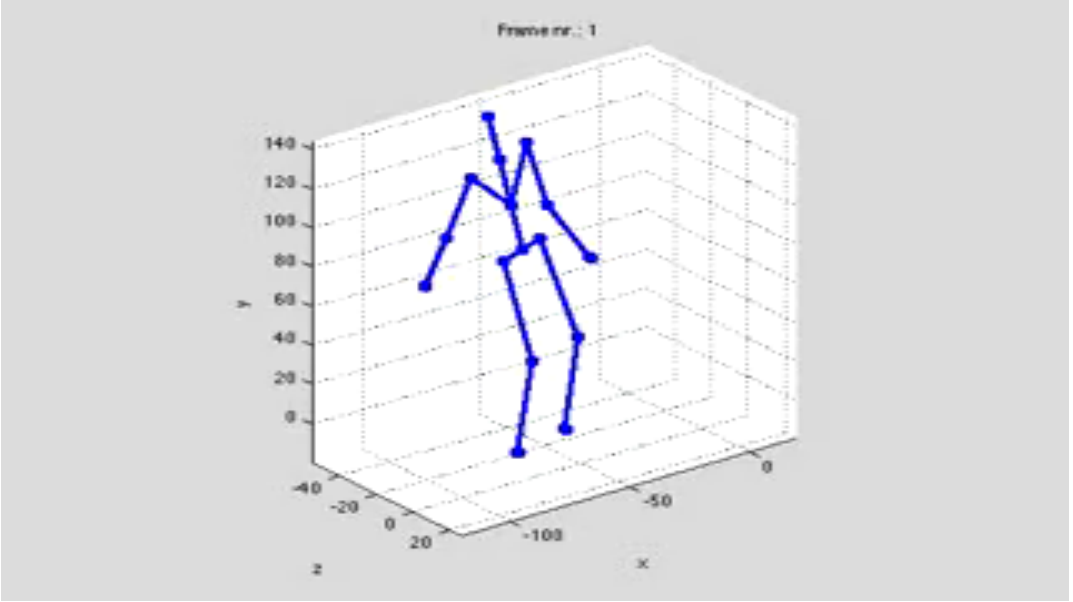
Graham Taylor


Filling in Missing Markers Using a Binary Latent Variable Model
Left leg
Upper body
001
The marker data for the left leg(left plot) and upper body(right plot) has been deleted from the test data at frame 63 onward.
A latent variable model with 400 binary stochastic units is used to fill in the missing data.
The model has been trained on 2646 frames of motion capture data, consisting of running and walking sequences.
Blue represents limbs for which the parent's joint angle data is known. Red represents data filled in by the model.
Graham Taylor

Ähnlich – Prädiktion von Bewegungen

- Fragestellung: Oft wiederholte Bewegungen lernen
  ➔
  Lernen Bewegungen zu generieren und und mit Vorlauf von x sec. vorherzusagen?

- Aufzeichnung mit Kinect1 (ungenau, großes Rauschen ...)

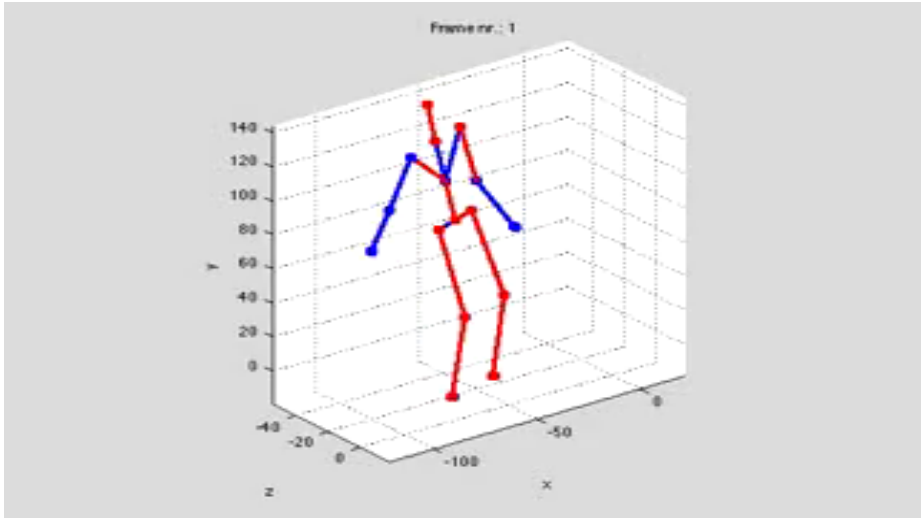


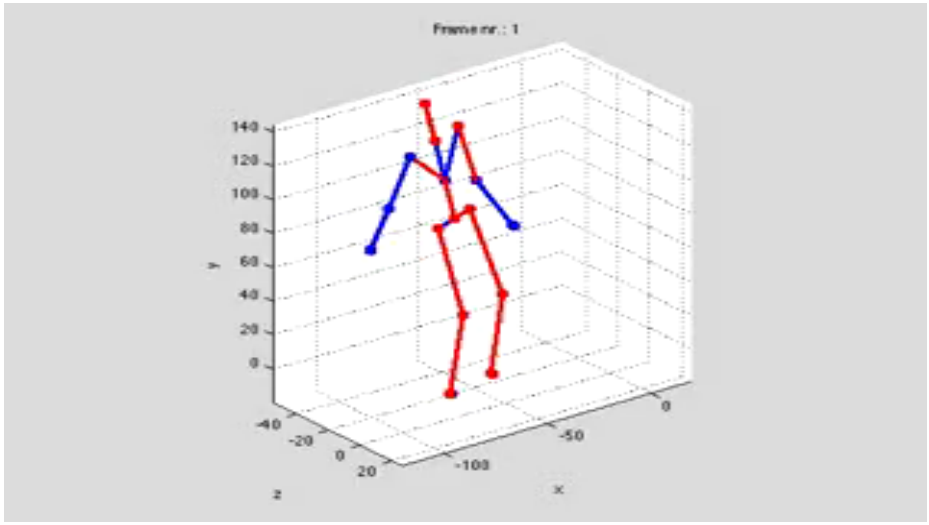
Erste Ergebnisse – Prädiktion von Bewegungen

- Lernen mit 2 - schichtigem temporalen RBMs
- Jeweils Anregung mit Daten → x Sek. Daten generieren (ohne weitere Eingabe - Daten) → rekonstruierte Daten ablesen

- Vorhersage rot, überlagert mit Original blau
  - Oben 2 sec.
  - Unten 6 sec.

- (kleine Datenmenge – kein ausgiebiger Test)





Literatur:

- Siehe **Geoffrey E. Hinton**

  http://www.cs.toronto.edu/~hinton/

  http://videolectures.net/mlss09uk_hinton_dbn/

- Marcus Frean: ACISS'09 tutorial on deep belief nets