

Basisfunktionen und Neuronale Netze

Volker Tresp

I am an AI optimist. We've got a lot of work in machine learning, which is sort of the polite term for AI nowadays because it got so broad that it's not that well defined.

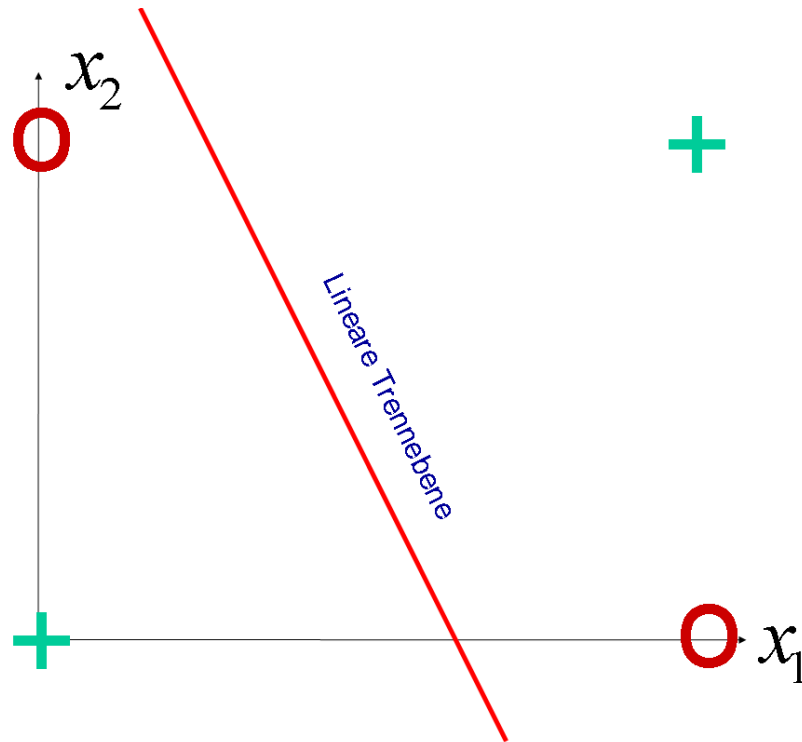
Bill Gates (Scientific American Interview, 2004)

"If you invent a breakthrough in artificial intelligence, so machines can learn," Mr. Gates responded, "that is worth 10 Microsofts." (Quoted in NY Times, Monday March 3, 2004)

Nichtlineare Abbildungen und Klassifikatoren

- Regression:
 - Es ist eher unwahrscheinlich, dass die wahre Funktion $f(\mathbf{x})$ linear ist, obwohl dies bei Problemstellungen in hohen Eingangsdimensionen durchaus eine praktische Annahme ist. (Oder in der Physik: $F = ma$)
 - Wir wollen die Mächtigkeit unseres Modelles erhöhen, so dass auch beliebige nicht-lineare funktionelle Abhängigkeiten gelernt werden können
- Klassifikation:
 - Ebenso kann man annehmen, dass lineare Trennflächen für die Mehrzahl der Anwendungen nicht optimal sind
 - Wir wollen die Mächtigkeit unseres Modelles erhöhen, so dass auch beliebige nicht-lineare Trennflächen modelliert werden können

XOR ist nicht linear separierbar

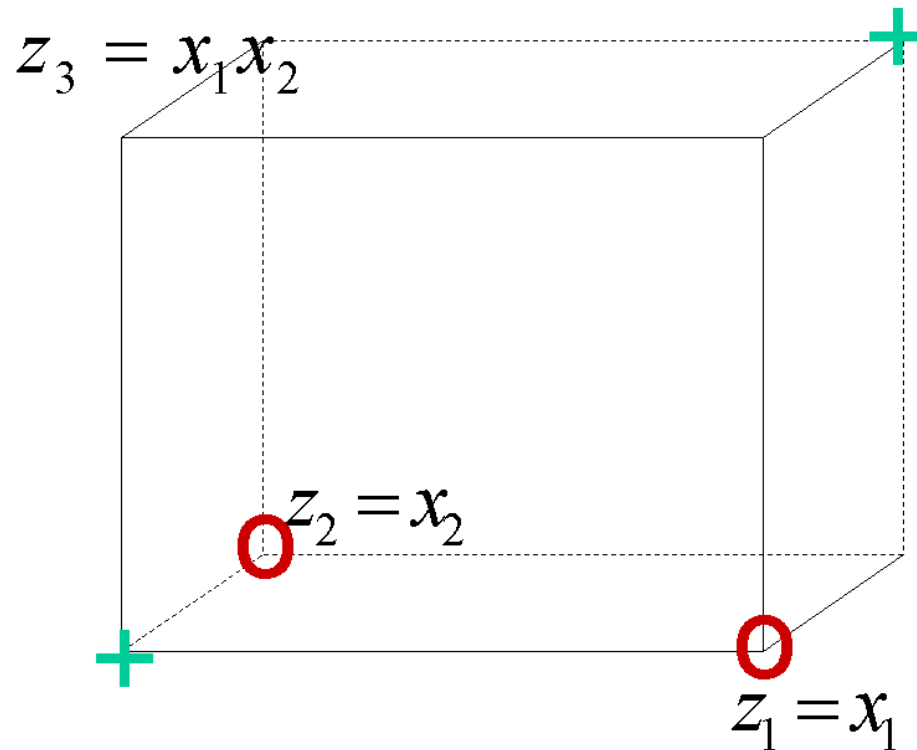


Wie kann man es dennoch schaffen, unter Zuhilfenahme eines linearen Klassifikators, eine nichtlineare Trennfläche zu realisieren?

Hinzunahme weiterer Basisfunktionen

- Lineares Modell: Eingangsvektor: $1, x_1, x_2$
- Lineares Modell mit zusätzlicher Basisfunktion: $1, x_1, x_2, x_1x_2$
- Der Wechselwirkungsterm (Interaktionsterm) x_1x_2 koppelt die Eingänge in einer nichtlinearen Weise

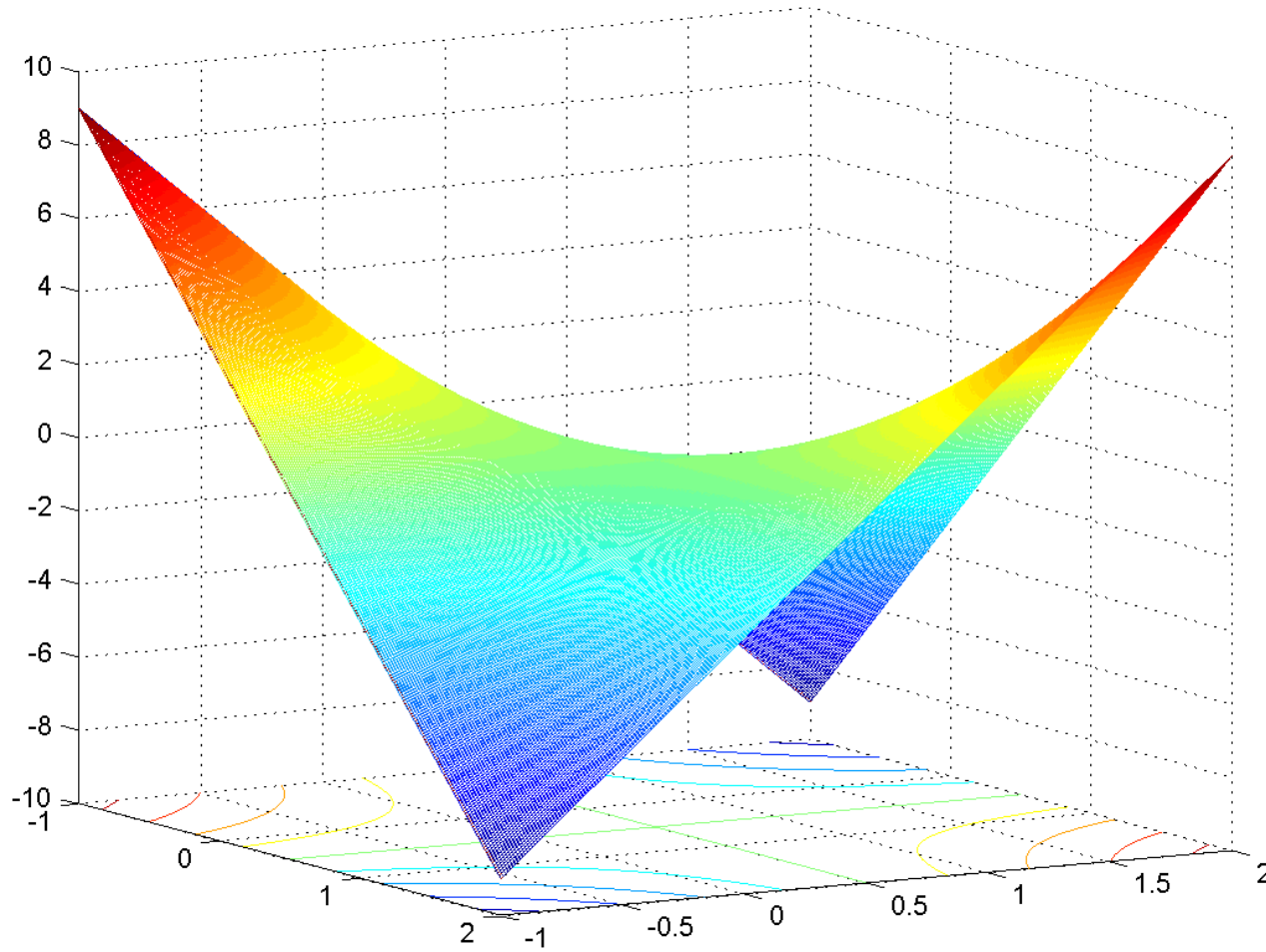
Mit $z_3 = x_1x_2$ wird das XOR linear separierbar



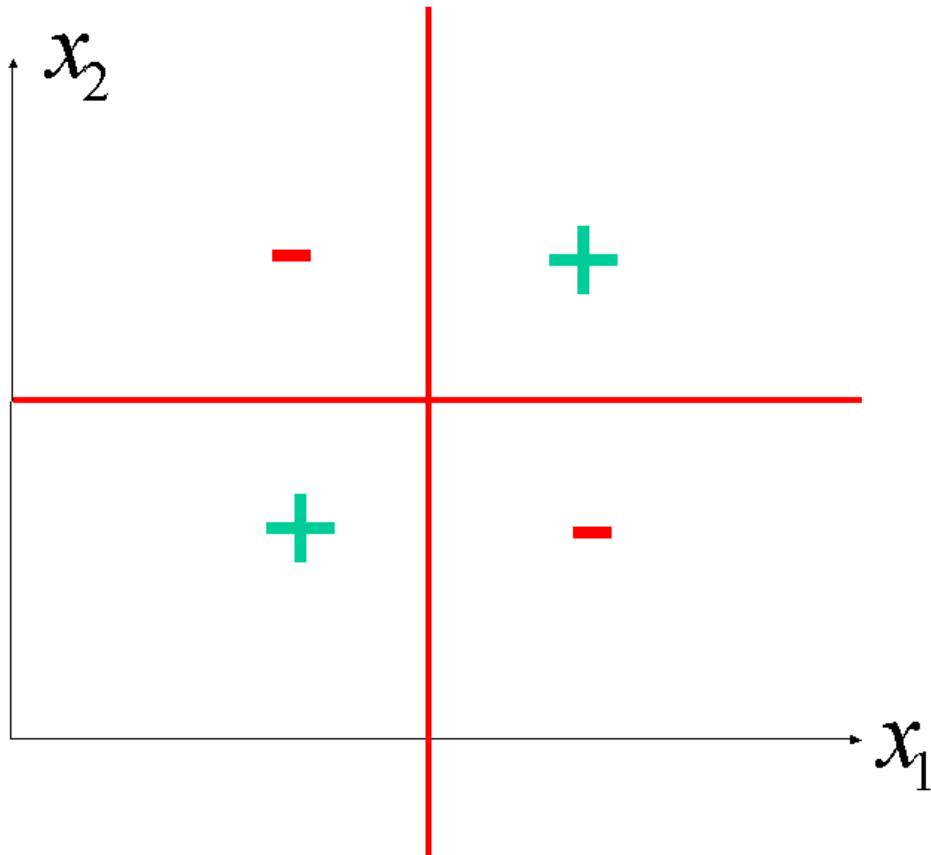
$$f(\mathbf{x}) = 1 - 2x_1 - 2x_2 + 4x_1x_2$$

$$\mathbf{z} = (1, x_1, x_2, x_1x_2) \text{ with } \mathbf{w} = (1, -2, -2, 4)$$

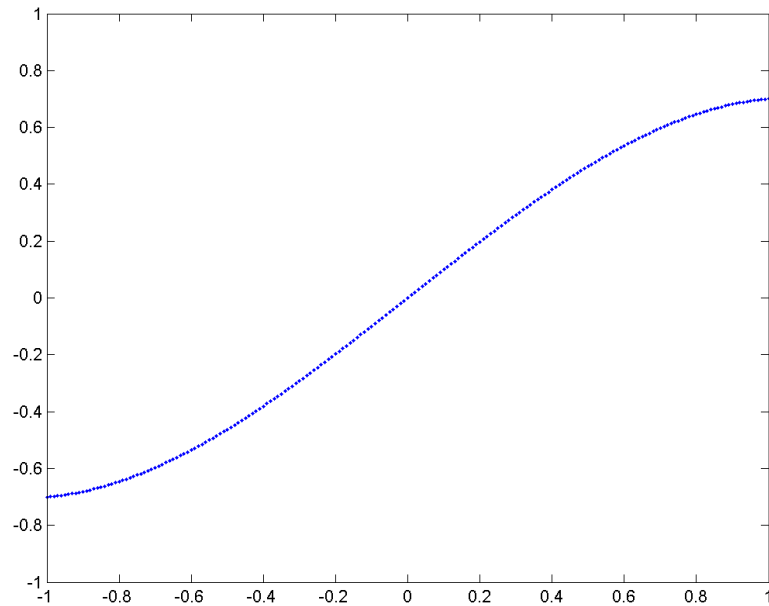
$$f(\mathbf{x}) = 1 - 2x_1 - 2x_2 + 4x_1x_2$$



Trennebenen

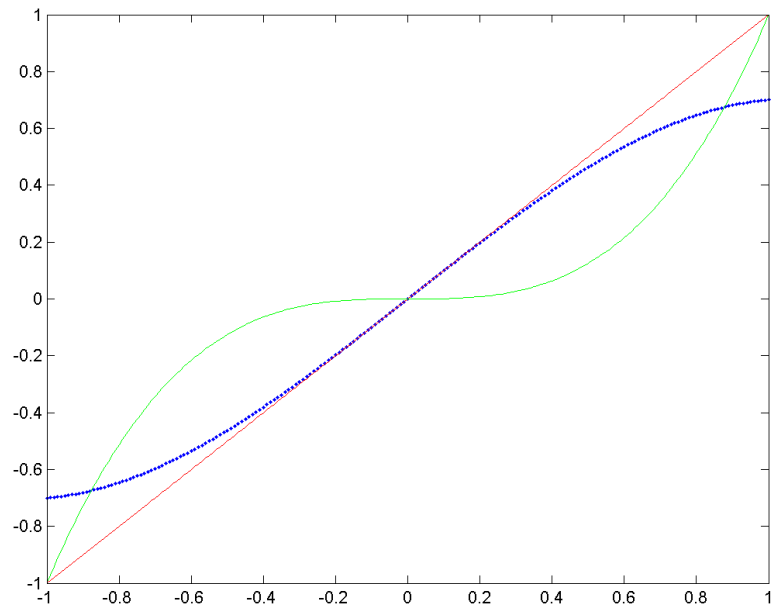


Eine Nichtlineare Abbildung



Wie kann man es schaffen, unter Zuhilfenahme einer linearen Regression, eine nichtlineare Abbildung zu realisieren?

$$f(x) = x - 0.3x^3$$



Basisfunktionen $z = (1, x, x^2, x^3)$ mit $w = (0, 1, 0, -0.3)$

Lineare Regression (Ridge Regression)

- Mehrdimensionales lineares Modell:

$$f(\mathbf{x}_i, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j x_{i,j} = \mathbf{x}_i^T \mathbf{w}$$

- Regularisierte Kostenfunktion

$$J_N^{pen}(\mathbf{w}) = \sum_{i=1}^N (y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \sum_{i=0}^M w_i^2$$

- Lösung

$$\hat{\mathbf{w}}_{Pen} = \left(\mathbf{X}^T \mathbf{X} + \lambda I \right)^{-1} \mathbf{X}^T \mathbf{y} \quad \text{mit} \quad \mathbf{X} = \begin{pmatrix} x_{1,0} & \cdots & x_{1,M-1} \\ \cdots & \cdots & \cdots \\ x_{N,0} & \cdots & x_{N,M-1} \end{pmatrix}$$

Lineare Regression mit dem Formalismus der Basisfunktionen

- Neu: $\phi_j(\mathbf{x}) = x_j$: Basisfunktion $\phi_j(\mathbf{x})$ ist identisch zur j -ten Eingangsdimension
- Mehrdimensionales lineares Modell:

$$f(\mathbf{x}_i, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\mathbf{x}_i)$$

- Regularisierte Kostenfunktion

$$J_N^{pen}(\mathbf{w}) = \sum_{i=1}^N (y_i - f(\mathbf{x}_i, \mathbf{w}))^2 + \lambda \sum_{i=0}^M w_i^2$$

- Lösung

$$\hat{\mathbf{w}}_{Pen} = \left(\Phi^T \Phi + \lambda I \right)^{-1} \Phi^T \mathbf{y} \quad \text{mit} \quad \Phi = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \dots & \phi_{M-1}(\mathbf{x}_1) \\ \dots & \dots & \dots \\ \phi_0(\mathbf{x}_N) & \dots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}$$

Grundidee

- Die Grundidee ist denkbar einfach: Neben den Eingangsvariablen x_i , formen wir zusätzliche Variablen, die sich als deterministische Funktionen der Eingangsvariablen berechnen lassen
- Beispiel: polynomiale Basisfunktionen

$$\{1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2\}$$

- Die neuen Variablen $z_0, \dots, z_{M_\phi-1}$ werden durch Basisfunktionen $\{\phi_h(\mathbf{x})\}_{h=0}^{M_\phi-1}$ berechnet
- Im Beispiel:

$$z_0 = \phi_0(\mathbf{x}) = 1 \quad z_1 = \phi_1(\mathbf{x}) = x_1 \quad z_5 = \phi_5(\mathbf{x}) = x_1x_3 \quad \dots$$

- Unabhängig von der Wahl der Basisfunktionen, wird die Regression mit den obigen Gleichungen (der linearen Regression) berechnet

Lineare Regression mit der Schreibweise der Transformierten Variablen Z

- Mehrdimensionales lineares Modell:

$$f(\mathbf{z}_i, \mathbf{w}) = w_0 + \sum_{j=1}^{M-1} w_j z_{i,j} = \mathbf{z}_i^T \mathbf{w}$$

- Regularisierte Kostenfunktion

$$J_N^{pen}(\mathbf{w}) = \sum_{i=1}^N (y_i - f(\mathbf{z}_i, \mathbf{w}))^2 + \lambda \sum_{i=0}^M w_i^2$$

- Lösung

$$\hat{\mathbf{w}}_{Pen} = \left(\mathbf{Z}^T \mathbf{Z} + \lambda I \right)^{-1} \mathbf{Z}^T \mathbf{y} \quad \text{mit} \quad \mathbf{Z} = \begin{pmatrix} z_{1,0} & \dots & z_{1,M-1} \\ \dots & \dots & \dots \\ z_{N,0} & \dots & z_{N,M-1} \end{pmatrix}$$

Konstruktion nichtlinearer Systeme für Regression und Klassifikation

- Im neuen Raum $z_0, \dots, z_{M_\phi-1}$ können wir nun alle bereits behandelten linearen Ansätze zur Klassifikation und Regression verwenden und erhalten Systeme zur nicht-linearen Klassifikation und Regression!

- Regression:

$$f(\mathbf{x}) = \sum_{h=0}^{M_\phi-1} w_h \phi_h(\mathbf{x})$$

- Klassifikation mit Diskriminantenfunktion:

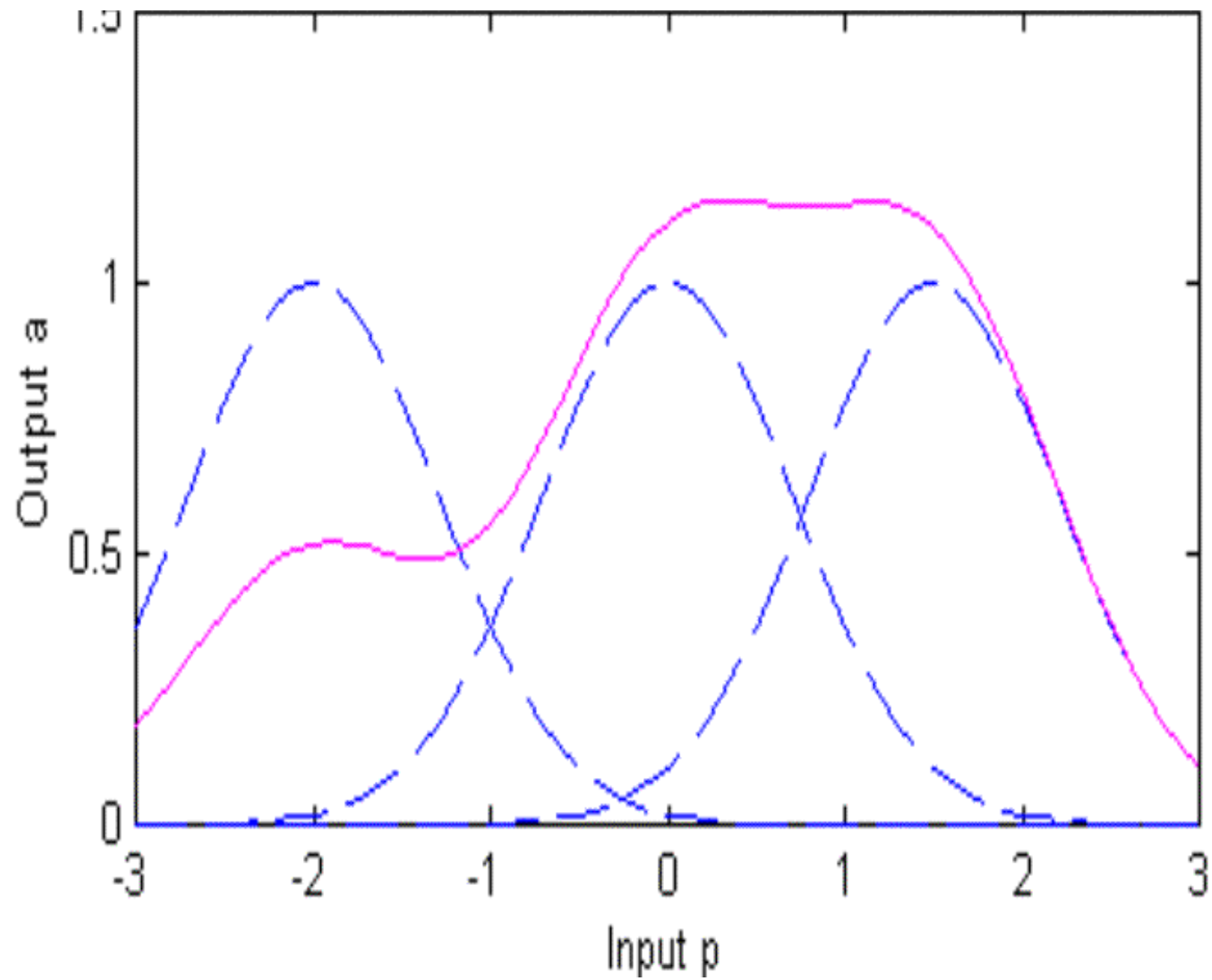
$$f(\mathbf{x}) = \sum_{h=0}^{M_\phi-1} w_h \phi_h(\mathbf{x})$$

Radiale Basisfunktionen (RBF)

- Wir hatten schon polynomiale Basisfunktionen kennengelernt
- Eine weitere beliebte Klasse von Basisfunktionen sind Radiale Basisfunktionen (RBF).
Typische Vertreter sind Gauss-Glocken

$$\phi_h(\mathbf{x}) = \exp\left(-\frac{1}{2s_h^2}|\mathbf{x} - \mathbf{c}_h|^2\right)$$

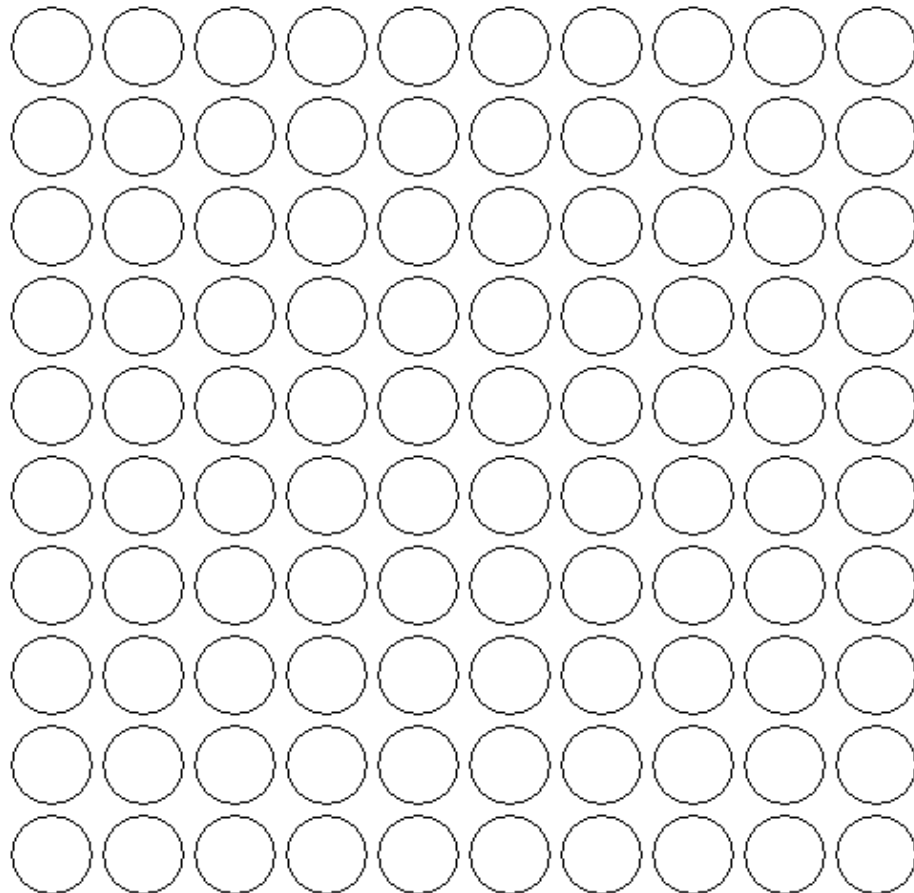
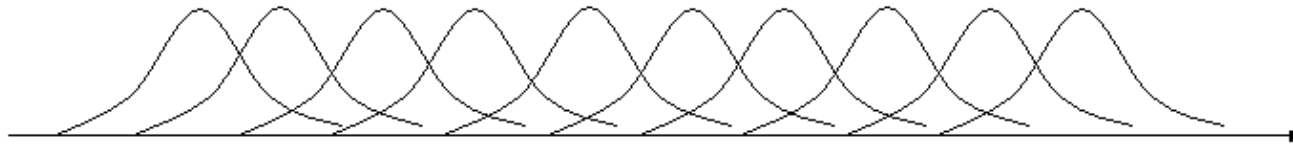
Drei RBFs (blau) formen $f(x)$ (pink)



Optimale Basisfunktionen

- Soweit schien alles etwas zu gut um wahr zu sein
- Hier ist der Pferdefuß: die Anzahl “sinnvoller” Basisfunktionen steigt exponentiell an mit der Anzahl der Eingangsvariablen
- Will ich z.B. K RBF's pro Dimension spendieren, benötige ich für M Dimensionen K^M RBFs
- Ähnlich schnell steigt die Anzahl sinnvoller Polynome mit der Dimensionalität
- *Die zentrale Frage zur Lösung nichtlinearer Probleme: wie erhalte ich eine kleine Anzahl problemangepasster Basisfunktionen*

10 RBFs in einer Dimension



100 RBFs in
zwei
Dimensionen

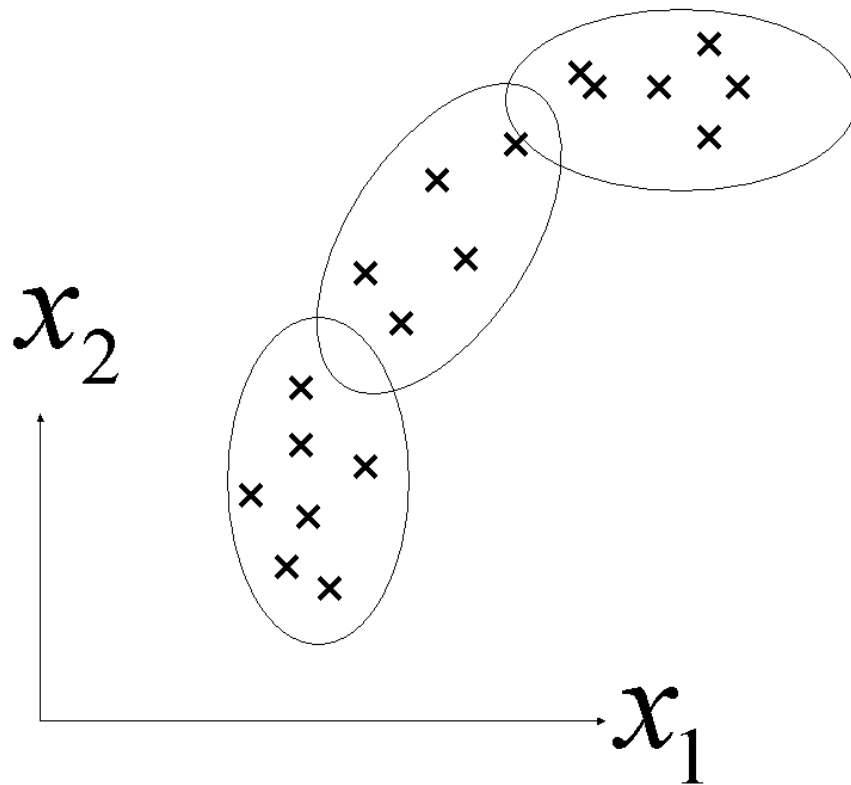
Modellselektion: Polynomiale Basisfunktionen

- Zunächst wird ein lineares Modell mit den Eingangsvariablen geformt
- Es werden polynomiale Basisfunktionen hinzugenommen und es wird geprüft, welche signifikant das Modell verbessern
- Besonders geeignet für Klassifikationsaufgaben (Polynomklassifikatoren: Siemens-Dematic OCR, J. Schürmann):
 - Dimensionsreduktion durch PCA
 - Begrenzte Dimensionserhöhung durch Einführung signifikanter Polynome
 - Lineare Klassifikation

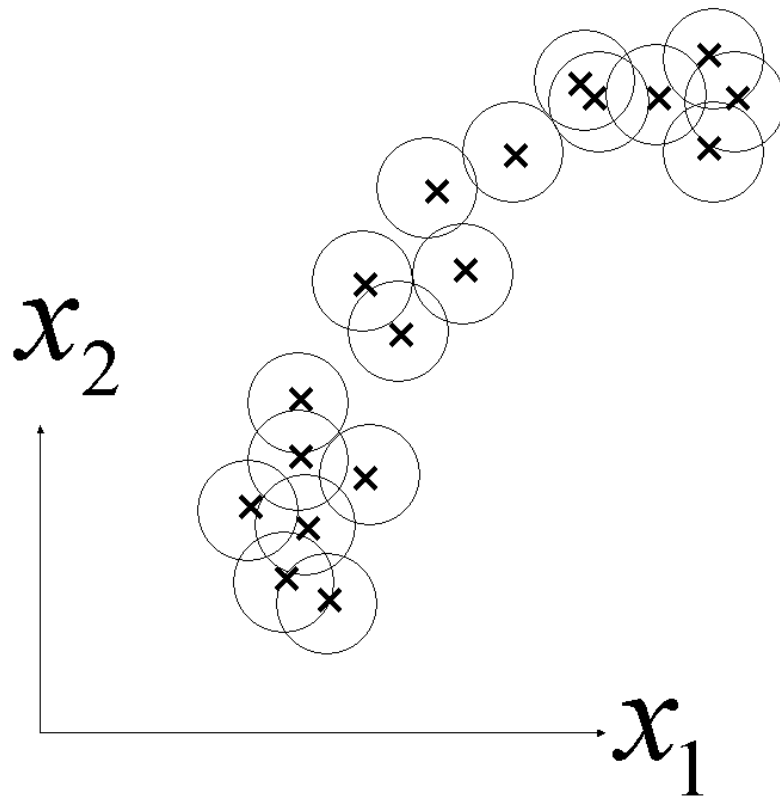
Modellselektion: RBFs

- Es kann sinnvoll sein, die Daten im Eingangsraum zu Gruppieren (Clustern) und die Clusterzentren als Zentren der Gaussglocken zu übernehmen
- Die Weiten der Gaussglocken ergeben sich z. B. dann aus der Varianz der Verteilung der Daten in den jeweiligen Clustern
- Ein weiteres sinnvolles Vorgehen besteht darin, soviele Gaussglocken zu definieren, wie es Datenpunkte gibt $M_\phi = N$. Die Zentren der Gaussglocken werden durch die Datenpunkte bestimmt, die Weiten der Gaussglocken werden über Kreuzvalidierung bestimmt (siehe Gauss Prozess Regression)

RBFs durch Clustern



RBFs für jeden Datenpunkt



Applikationsspezifische Merkmale

- Die Transformation der “Rohdaten” in eine applikationsspezifische Repräsentation stellt anwendungsspezifische Basisfunktionen dar; besonders wenn die Repräsentation hochdimensional ist
- Wir haben bereits die Vektorraumrepräsentation von Dokumenten kennengelernt
- Die Ableitung von oft einer großen Anzahl von Merkmalen für Dokumente, Bilder, Genesequenzen, ... ist ein sehr aktives Forschungsgebiet
- Mit anderen Worten: wenn die Vorverarbeitung schon eine hohe Zahl von Merkmalen liefert, dann ist die Wahrscheinlichkeit hoch, dass ein lineares System das Problem bereits lösen kann.
- Dies ist absolut bemerkenswert: Probleme können in hohen Dimensionen einfacher werden; vergleiche: Curse of Dimensionality (Bellman)

Flexiblere Modelle: Neuronale Netze

- Soweit wurden die Basisfunktionen (mehr oder weniger) als bekannt vorausgesetzt
- Auch bei einem Neuronalen Netz besteht der Ausgang aus der linear-gewichteten Summation von Basisfunktionen

$$f(x) = \sum_{h=0}^{M_\phi-1} w_h \phi_h(\mathbf{x}, \mathbf{v}_h)$$

Neuronale Netze: wesentliche Vorteile

- Neuronale Netze sind universelle Approximatoren: jede stetige Funktion kann beliebig genau approximiert werden (mit genügend vielen sigmoiden Basisfunktionen)
- Entscheidender Vorteil Neuronaler Netze: mit **wenigen** Basisfunktionen einen sehr guten Fit zu erreichen bei gleichzeitiger hervorragender Generalisierungseigenschaften
- Besondere Approximationseigenschaften Neuronaler Netze in hohen Dimensionen
- Basisfunktionen werden problemangepasst optimiert

Neuronale Netze: wesentliche Vorteile (2)

- Spezielle Form der Basisfunktionen

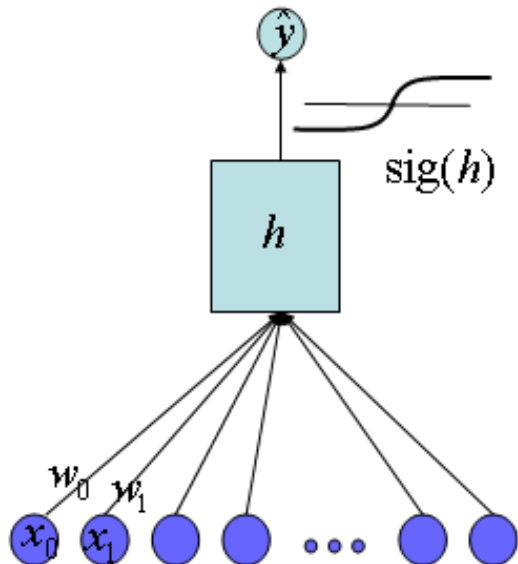
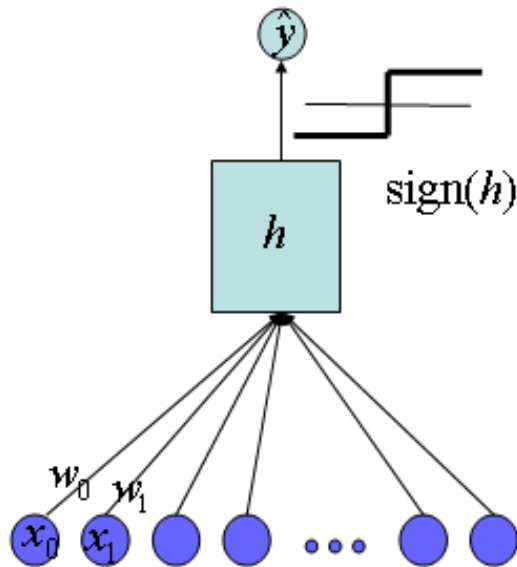
$$\phi_h(\mathbf{x}, \mathbf{v}_h) = z_h = \text{sig} \left(\sum_{j=0}^M v_{h,j} x_j \right)$$

mit

$$\text{sig}(in) = \frac{1}{1 + \exp(-in)}$$

- Adaption der inneren Parameter $v_{h,j}$ der Basisfunktionen!

Harte und weiche (sigmoide) Übertragungsfunktion



- Zunächst wird die Aktivierungsfunktion als gewichtete Summe der Eingangsgrößen x_i berechnet zu

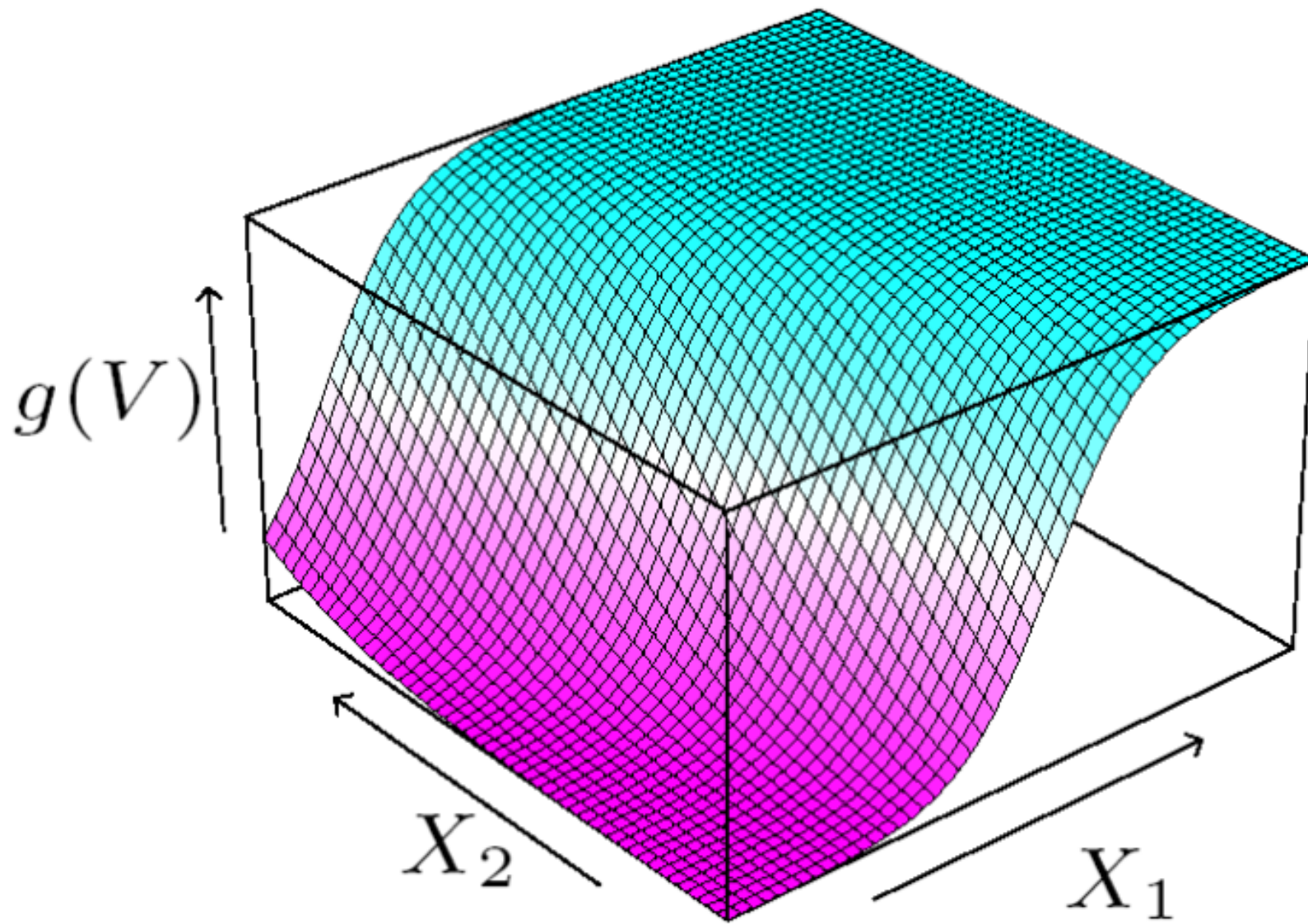
$$h = \sum_{j=0}^{M-1} w_j x_j$$

(beachte: $x_0 = 1$ ist ein konstanter Eingang, so dass w_0 dem Bias entspricht)

- Das sigmoide Neuron unterscheidet sich vom Perceptron durch die Übertragungsfunktion

$$\text{Perceptron : } \hat{y} = \text{sign}(h)$$

$$\text{Sigmoides Neuron : } \hat{y} = \text{sig}(h)$$



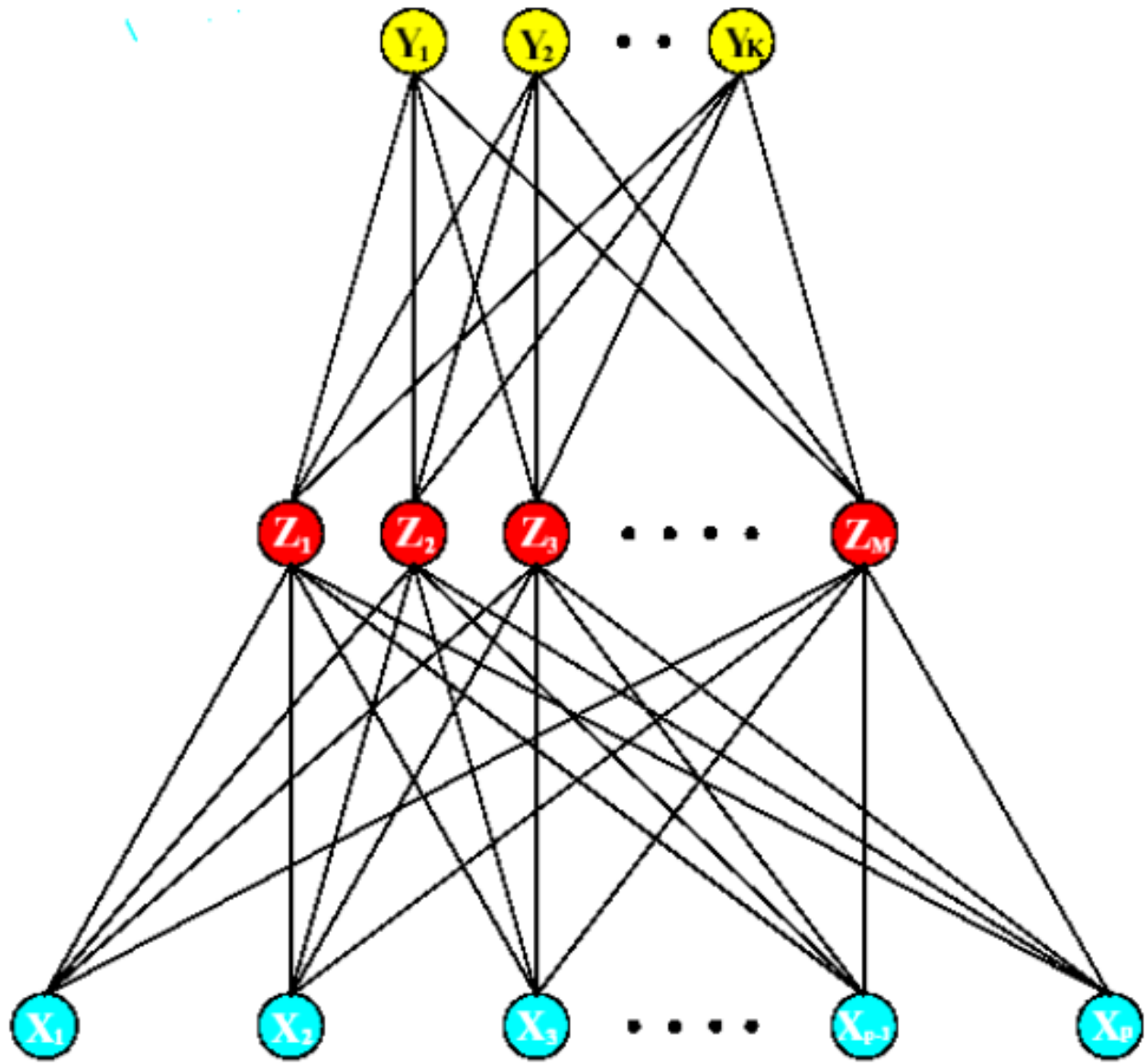
Transferfunktion

- Definiere die Hyperebene

$$z_h = \text{sig} \left(\sum_{j=0}^M v_{h,j} x_j \right) = 0.5$$

$$\sum_{j=0}^M v_{h,j} x_j = 0$$

- “Teppich über einer Stufe”



Lernen der Gewichte des Neuronales Netzes

- In der Regression gibt es ein lineares Ausgangsneuron und Ziel ist, ebenso wie bei der linearen Regression, die Minimierung des quadratischen Fehlers

$$J_N(\mathbf{w}, \mathbf{v}) = \sum_{i=1}^N (y_i - f(\mathbf{x}_i, \mathbf{w}, \mathbf{v}))^2$$

- Obwohl die logistische Regression auch populär ist, benutzt man für die Klassifikation ebenso die Minimierung des quadratischen Fehlers; für C -Klassen Klassifikation gibt es entsprechend C Ausgangsneuronen;
- Im folgenden beschäftigen wir uns mit der Minimierung des quadratischen Fehlers bezogen auf ein Ausgangsneuron

Gradientenbasiertes Lernen der Ausgangsgewichte des Neuronalen Netzes

- Fast alle in Frage kommenden Optimierungsverfahren benötigen den Gradienten der Kostenfunktion nach den Parametern. Für die Ausgangsgewichte ergibt sich:

-

$$\frac{\partial J_N(\mathbf{w}, \mathbf{v})}{\partial w_h} = -2 \sum_{i=1}^N z_h(\mathbf{x}_i) (y_i - f(\mathbf{x}_i, \mathbf{w}, \mathbf{v}))$$

- Die Optimierung der Ausgangsparameter \mathbf{w} ist einfach (vergleiche: ADALINE Lernregel)
- Musterbasierter Gradientenabstieg: $w_h \leftarrow w_h + \eta z_h(\mathbf{x}_i) (y_i - f(\mathbf{x}_i, \mathbf{w}))$

Lernen der Eingangsgewichte des Neuronalen Netzes

- Die Optimierung der Eingangsparameter $\mathbf{v} = (v_{1,1}, v_{1,2}, \dots, v_{M_\phi-1, M})^T$
- Die Kostenfunktion ist: nichtkonvex, nicht-quadratisch, nichtlinear und besitzt eine grosse Anzahl lokaler Minima
- Der Gradient berechnet sich zu (Kettenregel, Backpropagation Regel)

$$\frac{\partial J_N(\mathbf{w}, \mathbf{v})}{\partial v_{h,j}} = -2 \sum_{i=1}^N w_h z'_h(\mathbf{x}_i) x_{i,j} (y_i - f(\mathbf{x}_i, \mathbf{w}, \mathbf{v}))$$

$$z'_h = \frac{\exp(-z_h)}{(1 + \exp(-z_h))^2} = z_h(1 - z_h)$$

Backprop Lernregel

- Batch Mode

$$v_{h,j}(t + 1) = v_{h,j}(t) - \eta \frac{\partial J_N(\mathbf{w}, \mathbf{v})}{\partial v_{h,j}}$$

- Musterbasierter Gradientenabstieg mit Trainingsmuster (\mathbf{x}_i, y_i)

$$v_{h,j} \leftarrow v_{h,j} + \eta w_h z_h(\mathbf{x}_i)(1 - z_h(\mathbf{x}_i)) x_{i,j} [y_i - f(\mathbf{x}_i, \mathbf{w}, \mathbf{v})]$$

- Quasi Newton, Konjugiertes Gradientenverfahren, ...

Neuronale Netze: Variationen

- Mehrere Ausgänge (z. B. Klassifikation)
- Sigmoidale Transformation auch am Ausgangsneuron
- Mehrere versteckte Schichten

Neuronale Netze: Regularisierung

- Einfügen und Optimierung eines Regularisierungsterms

$$J_N^{pen}(\mathbf{w}, \mathbf{v}) = \sum_{i=1}^N (y_i - f(\mathbf{x}_i, \mathbf{w}, \mathbf{v}))^2 + \lambda_1 \sum_{h=0}^{M_\phi-1} w_h^2 + \lambda_2 \sum_{h=0}^{M_\phi-1} \sum_{j=0}^M v_{h,j}^2$$

$$\frac{\partial J_N^{pen}(\mathbf{w}, \mathbf{v})}{\partial w_h} = \frac{\partial J_N(\mathbf{w}, \mathbf{v})}{\partial w_h} + 2\lambda_1 w_h$$

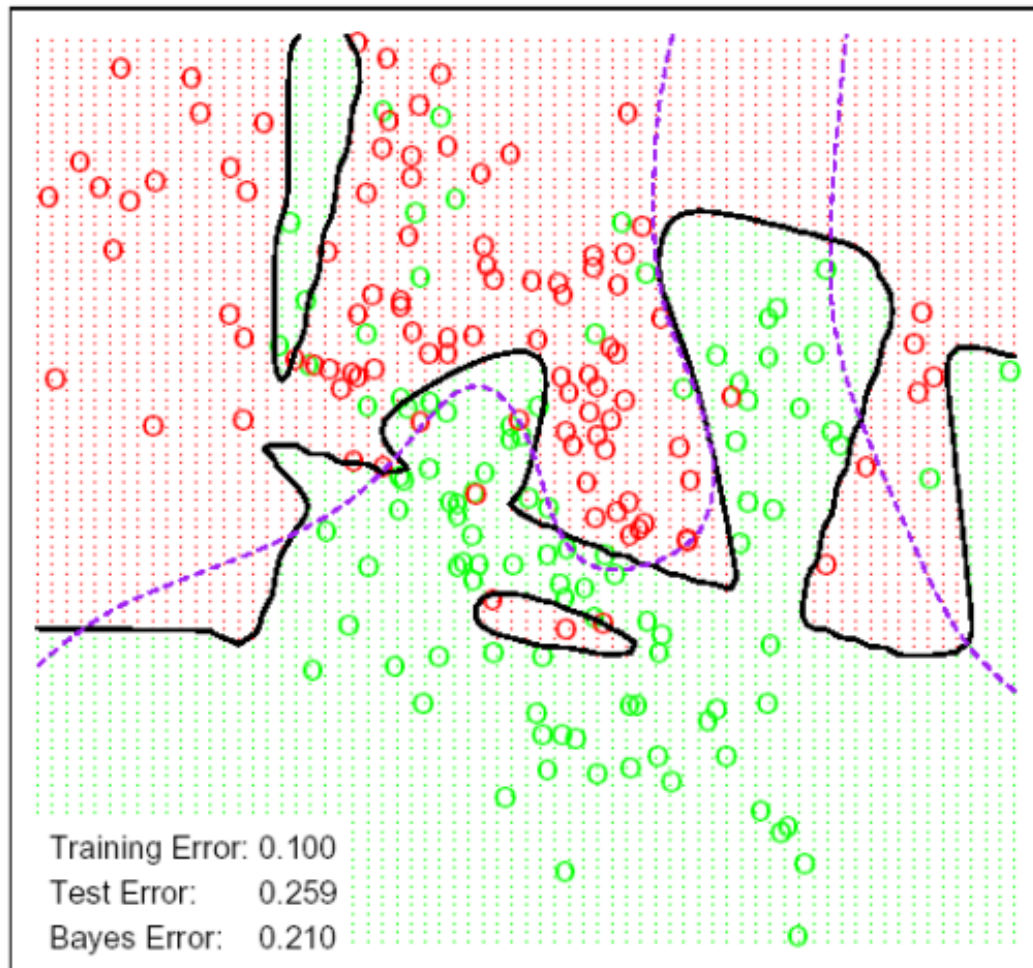
$$\frac{\partial J_N^{pen}(\mathbf{w}, \mathbf{v})}{\partial v_{h,j}} = \frac{\partial J_N(\mathbf{w}, \mathbf{v})}{\partial v_{h,j}} + 2\lambda_2 v_{h,j}$$

- Stopped Training

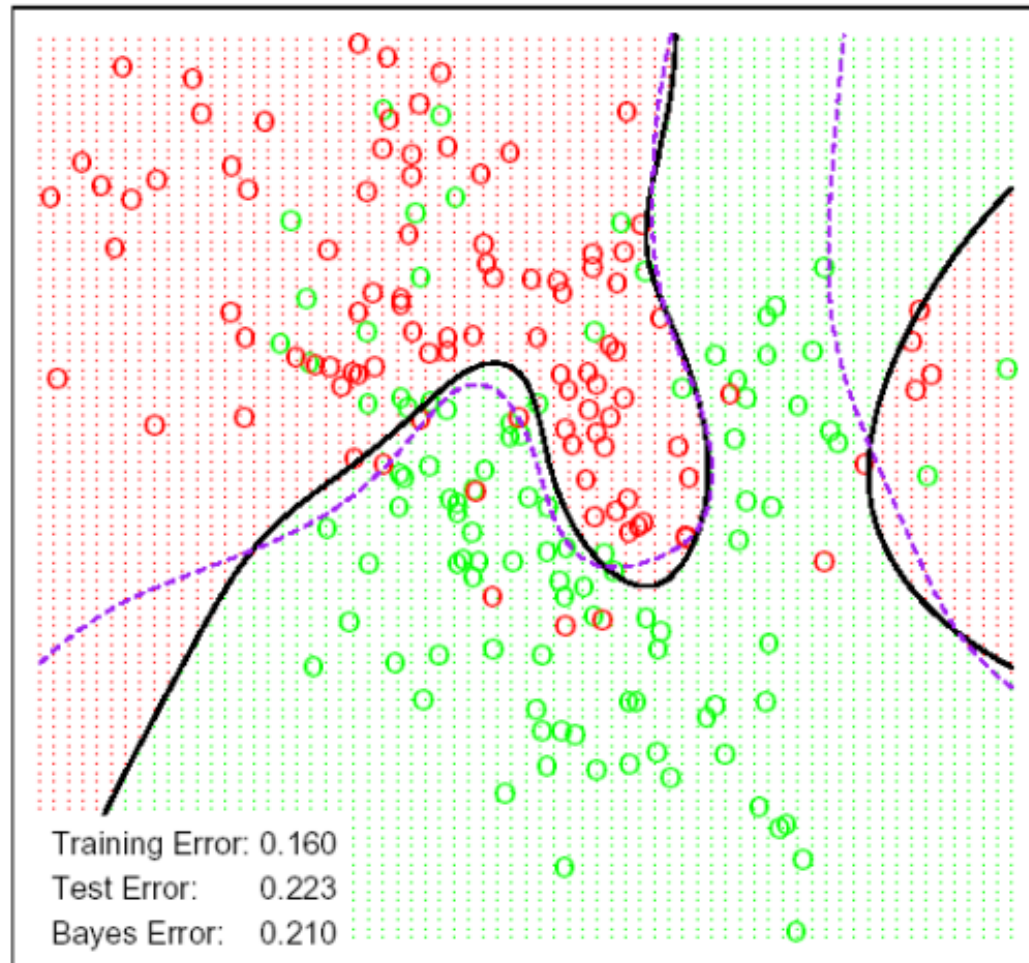
Musterbasiertes Lernen mit Regularisierung (Weight Decay)

- Trainingsmuster (\mathbf{x}_i, y_i)
- $w_h \leftarrow w_h + \eta_1 z_h(\mathbf{x}_i) (y_i - f(\mathbf{x}_i, \mathbf{w})) - \eta_2 w_h$
- $v_{h,j} \leftarrow v_{h,j} + \eta_1 w_h z_h(\mathbf{x}_i)(1 - z_h(\mathbf{x}_i)) x_{i,j} [y_i - f(\mathbf{x}_i, \mathbf{w}, \mathbf{v})] - \eta_2 v_{h,j}$

Neural Network - 10 Units, No Weight Decay



Neural Network - 10 Units, Weight Decay=0.02



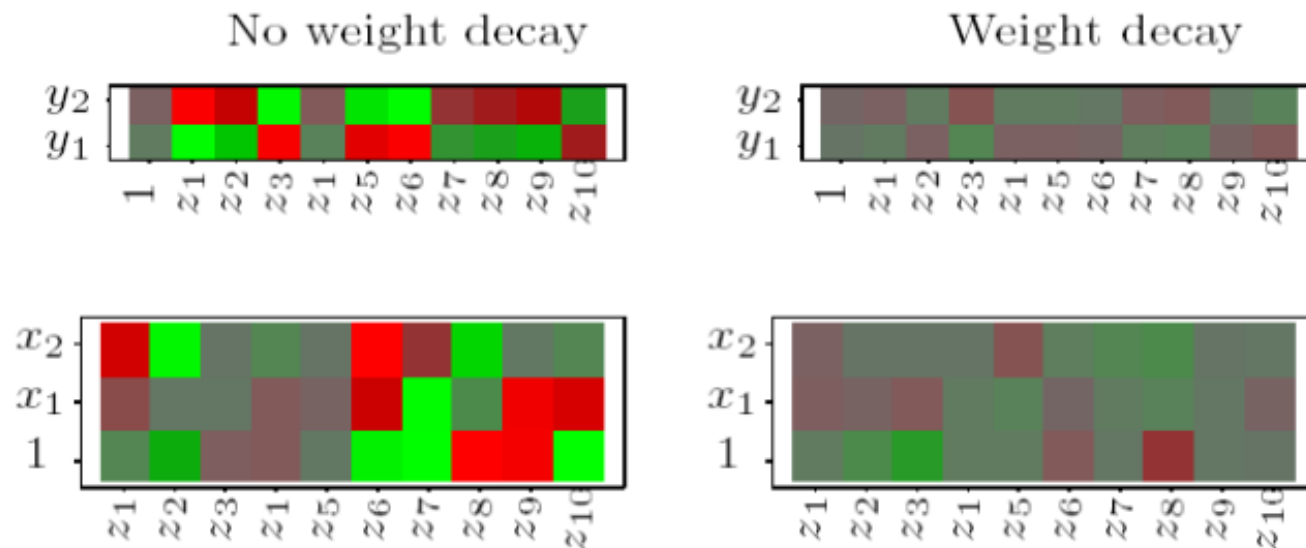
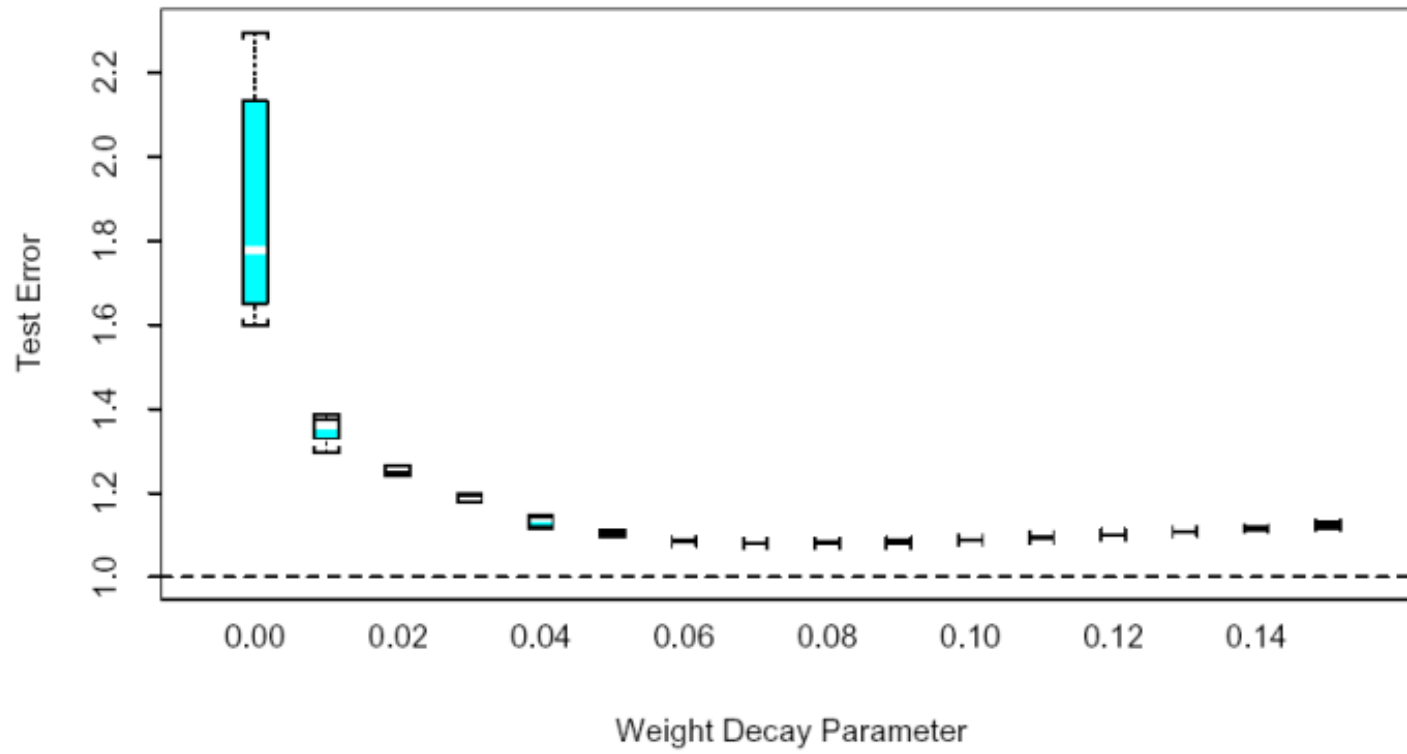
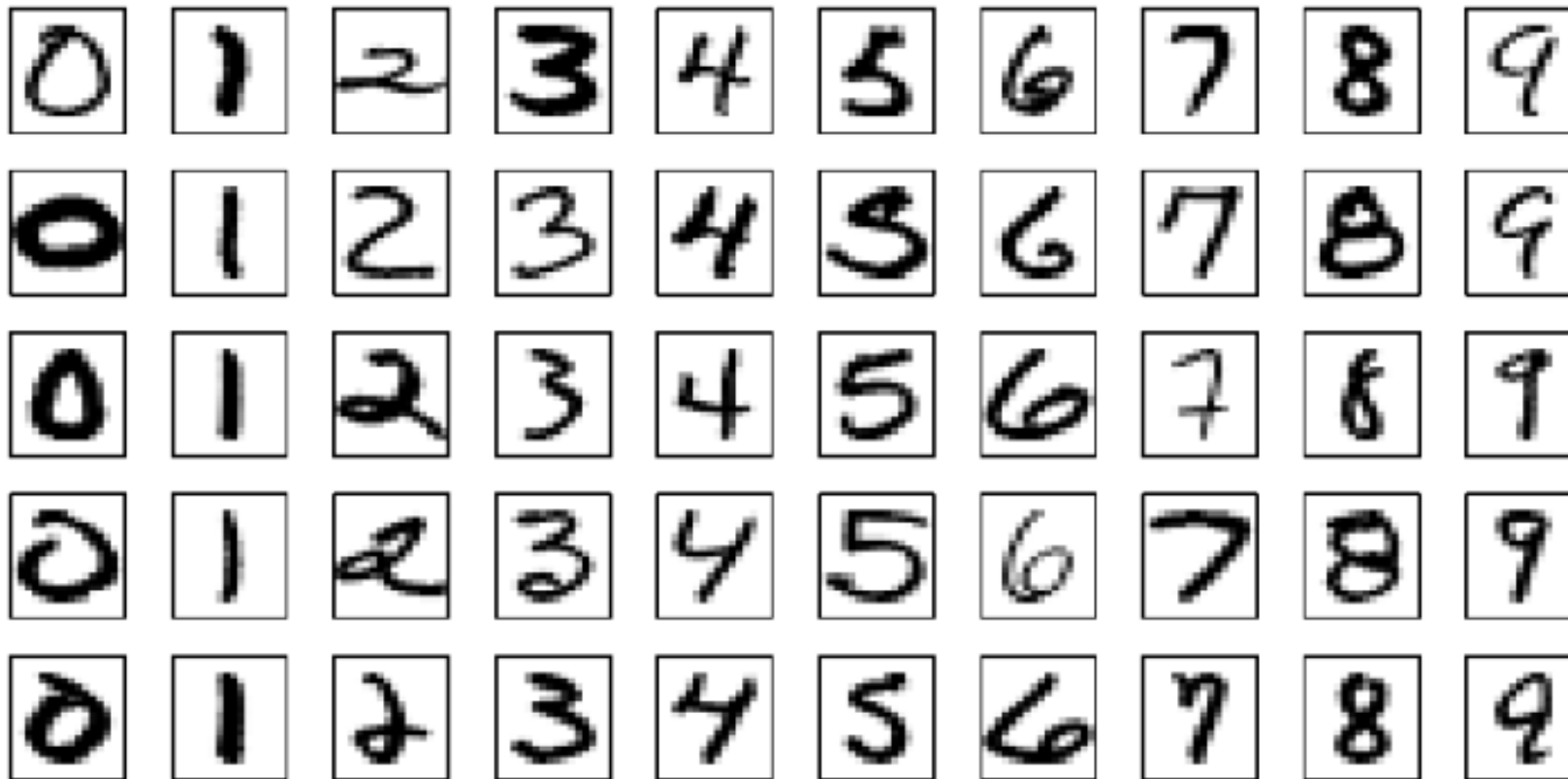


Figure 11.5: *Heat maps of the estimated weights from the training of neural networks from Figure 5. The display ranges from bright green (negative) to bright red (positive).*

Sum of Sigmoids, 10 Hidden Unit Model

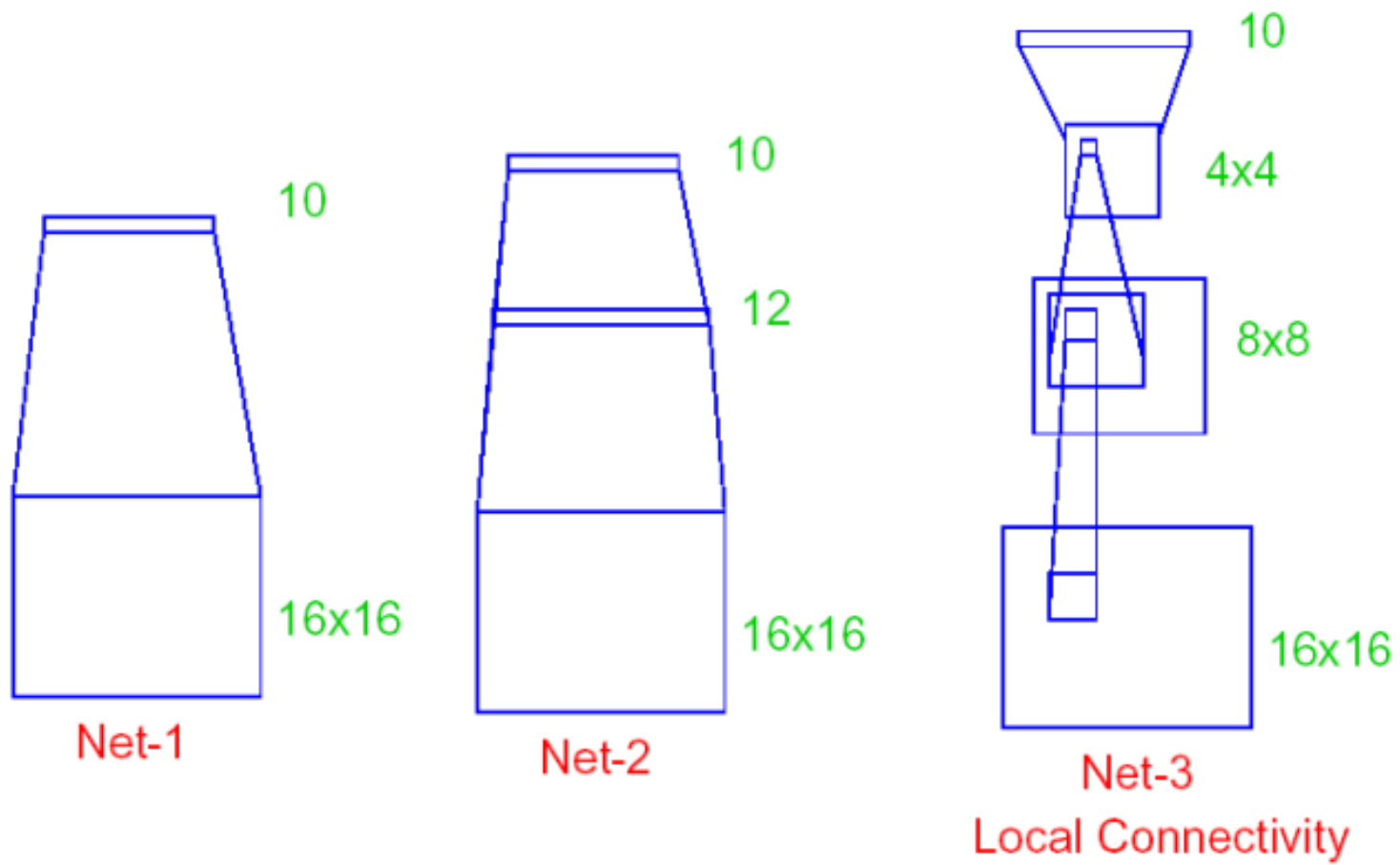


OCR Application



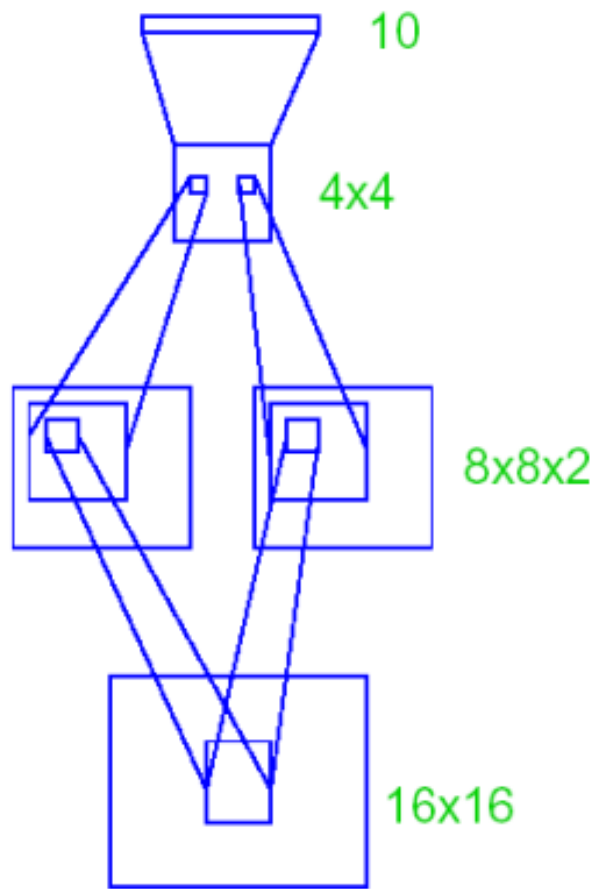
OCR (Bell Labs)

- Im Beispiel hier: 16×16 grauwertige Bilder; 320 Ziffern im Trainingssatz, 160 Ziffern im Testsatz
- Net-1: Keine versteckte Schicht: entspricht in etwa logistischer Regression (10 Perzeptrons)
- Net-2: Eine versteckte Schicht mit 12 Knoten; vollvernetzt (normales Neuronales Netz mit einer versteckten Schicht)
- Net-3: Zwei versteckte Schichten mit lokaler Verbindungsstruktur (ohne weight-sharing!)
 - Jede der 8×8 Neuronen in der ersten versteckten Schicht sind nur mit 3×3 Eingangsneuronen verbunden aus einem rezeptivem Feld verbunden
 - In der zweiten versteckten Schicht ist jedes der 4×4 Neuronen mit 5×5 Neuronen der ersten versteckten Schicht verbunden
 - Net-3 hat weniger als 50% der Gewichte als Net-2 aber mehr Neuronen



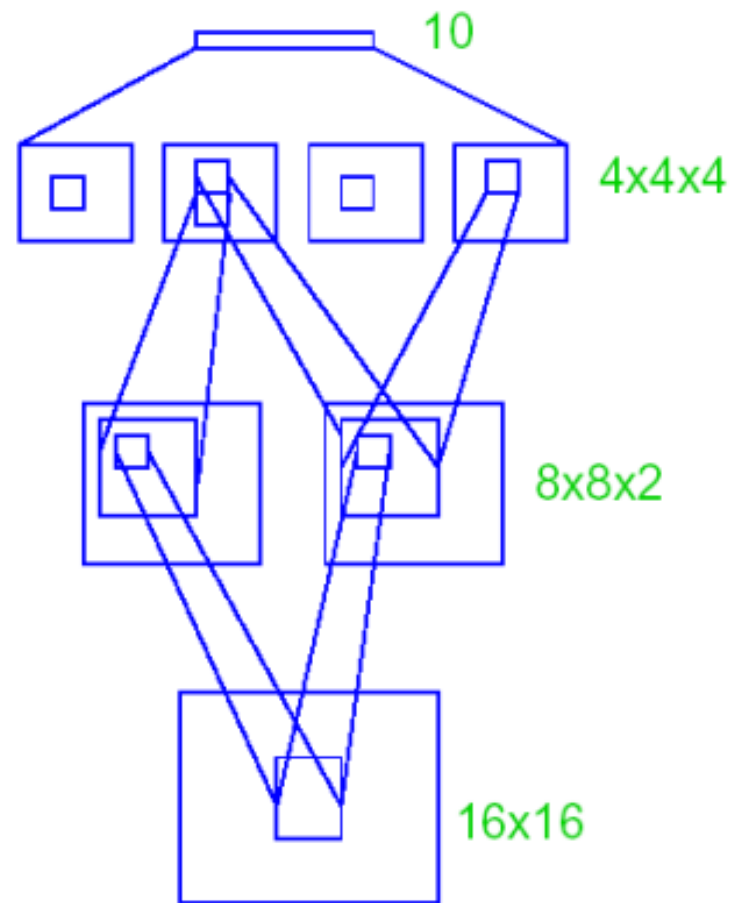
Weitere LeNets

- Net-4: Zwei versteckte Schichten mit lokaler Verbindungsstruktur und weight-sharing
- Net-5: Zwei versteckte Schichten mit lokaler Verbindungsstruktur und zwei Ebenen von weight-sharing
- Trainingsfehler für alle Netze gleich 0 (da sehr viele freie Parameter)



Net-4

Shared Weights



Net-5

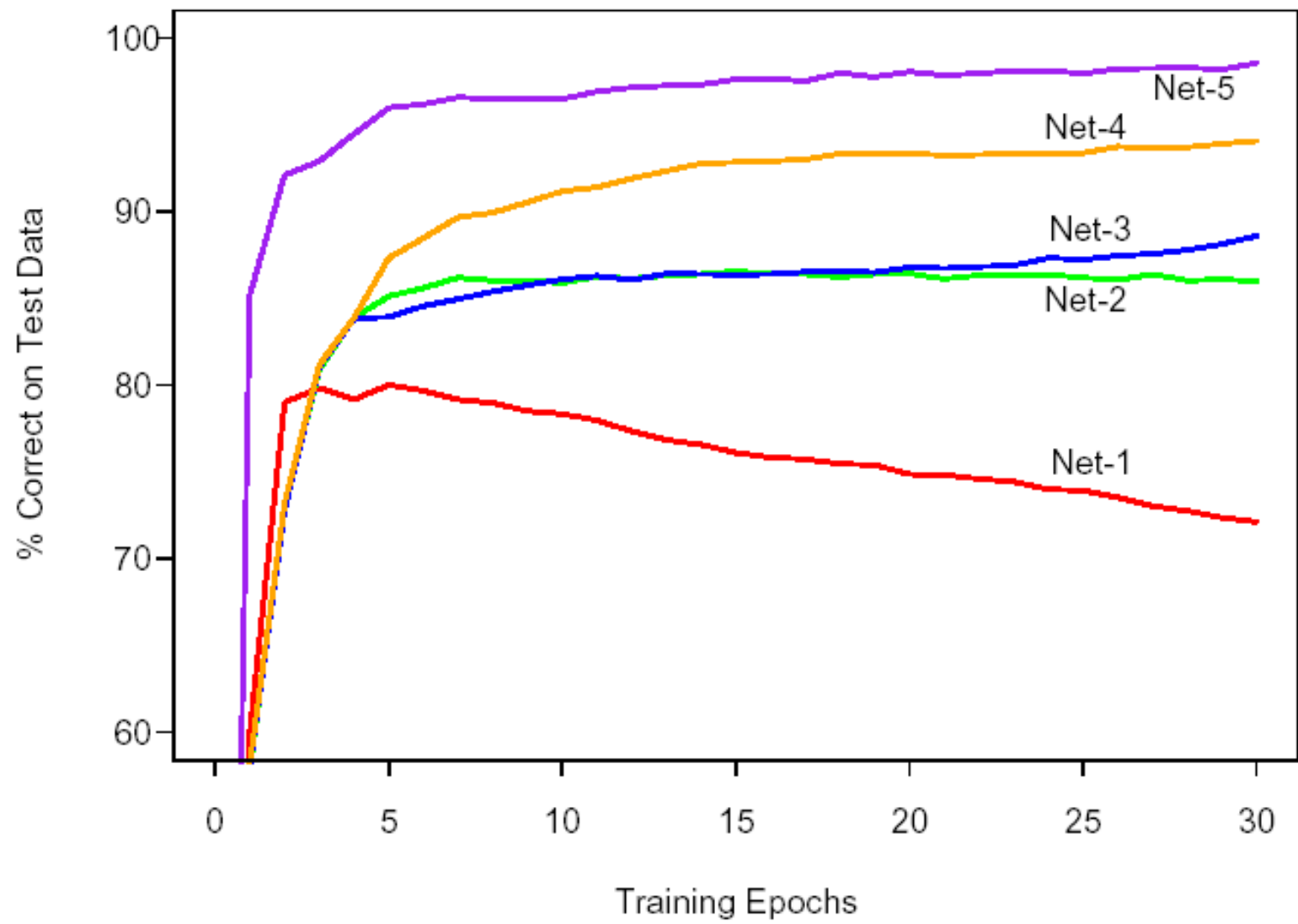
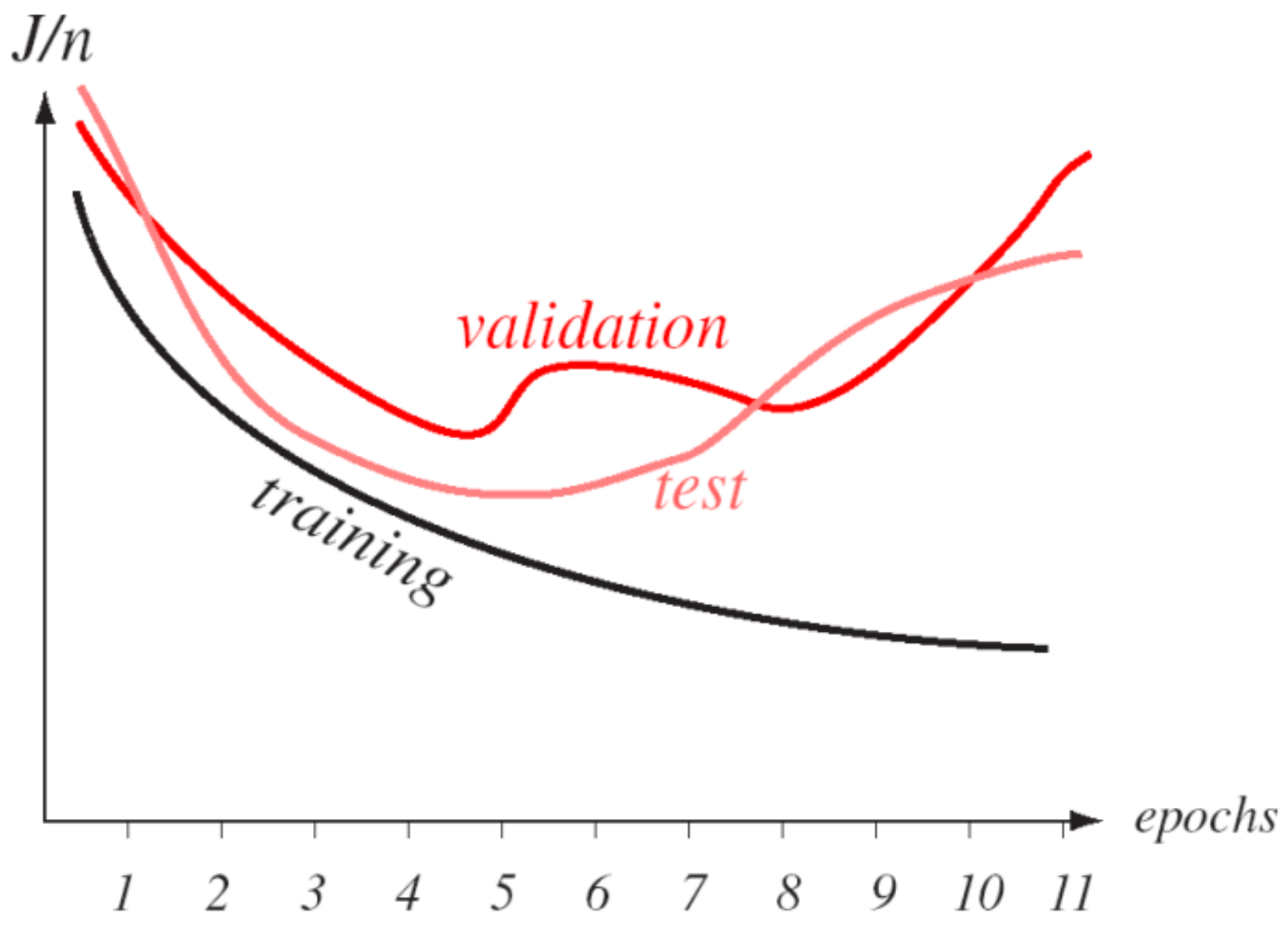


TABLE 11.1. *Test set performance of five different neural networks on a handwritten digit classification example (Le Cun, 1989).*

	Network Architecture	Links	Weights	% Correct
Net-1:	Single layer network	2570	2570	80.0%
Net-2:	Two layer network	3214	3214	87.0%
Net-3:	Locally connected	1226	1226	88.5%
Net-4:	Constrained network 1	2266	1132	94.0%
Net-5:	Constrained network 2	5194	1060	98.4%

OCR: Kommentare

- Der reale Datensatz ist entsprechend größer
- Hier haben standard Neuronale Netze eine Fehlerrate von 4.5 % erreicht (*vanilla back-prop*)
- *LeNet-5* ist eine komplexere Version von Net-5 und erreicht eine Fehlerrate von nur 0.8 %



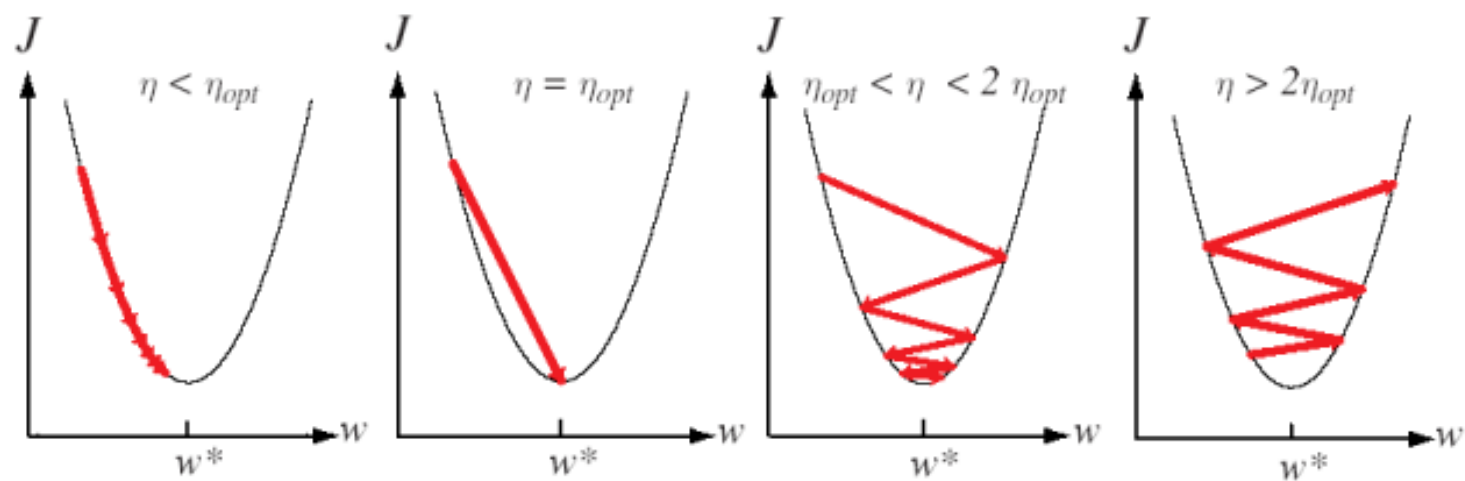
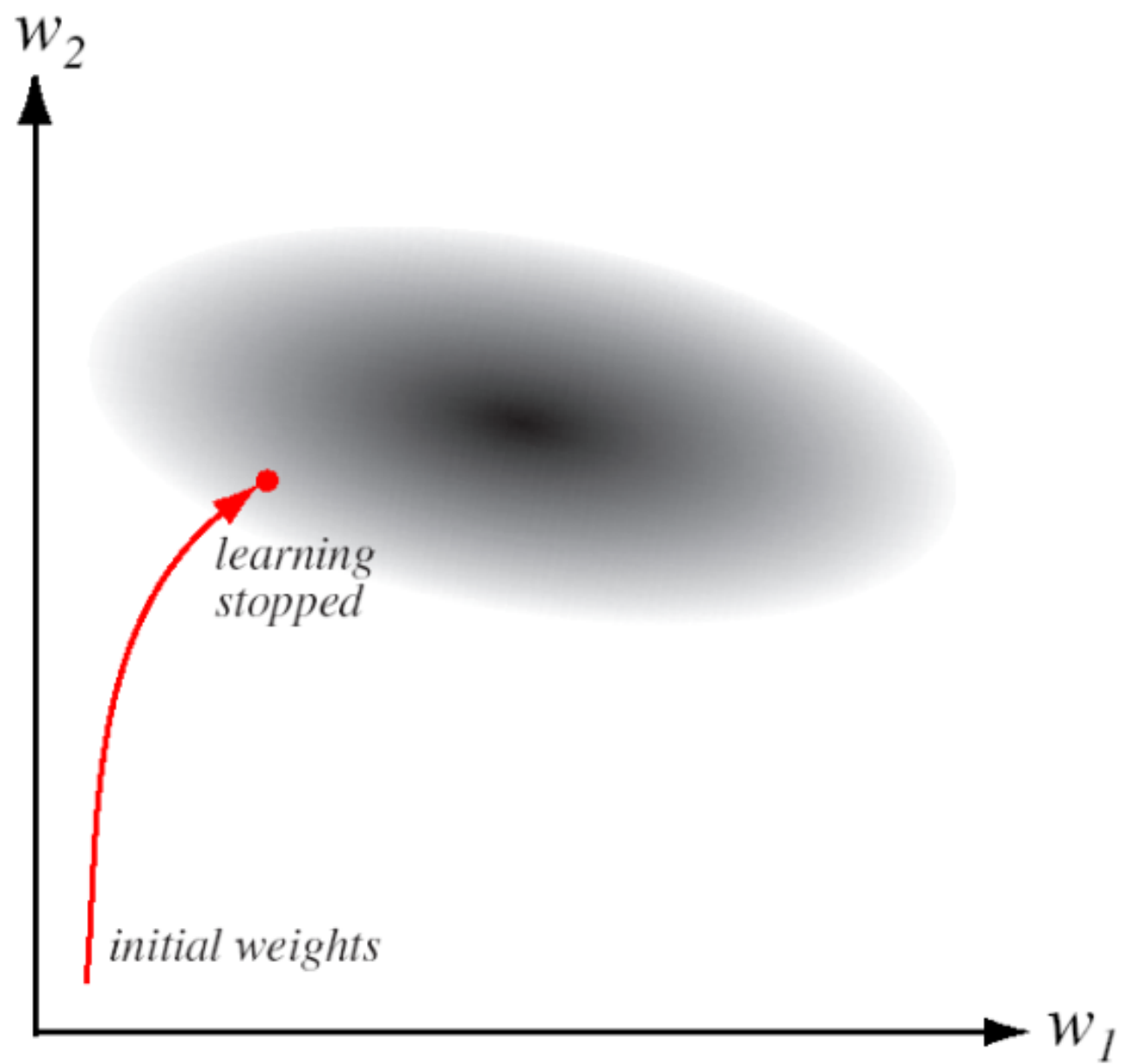


FIGURE 6.16. Gradient descent in a one-dimensional quadratic criterion with different learning rates. If $\eta < \eta_{opt}$, convergence is assured, but training can be needlessly slow. If $\eta = \eta_{opt}$, a single learning step suffices to find the error minimum. If $\eta_{opt} < \eta < 2\eta_{opt}$, the system will oscillate but nevertheless converge, but training is needlessly slow. If $\eta > 2\eta_{opt}$, the system diverges. From: Richard O. Duda, Peter E. Hart, and David G. Stork, *Pattern Classification*. Copyright © 2001 by John Wiley & Sons, Inc.



Zusammenfassung

- Neuronale Netze sind sehr leistungsfähig mit hoher Performanz
- Das Training ist aufwendig aber man kann mit einiger Berechtigung sagen, dass das Neuronale Netz tatsächlich etwas “lernt”, nämlich die optimale Repräsentation der Eingangsdaten in der versteckten Schicht
- Die Vorhersage ist schnell berechenbar
- Neuronale Netze sind universelle Approximatoren
- Neuronale Netze besitzen sehr gute Approximationseigenschaften
- Nachteil: das Training hat etwas von einer Kunst; eine Reihe von Größen müssen bestimmt werden (Anzahl der versteckten Knoten)
- Nachteil: Ein trainiertes Netz stellt ein lokales Optimum dar; Nichteindeutigkeit der Lösung
- Aber: In einem neuen Benchmarktest hat ein (Bayes'sches) Neuronales Netz alle konkurrierenden Ansätze geschlagen!

Neuronale Netze in der Zeitreihenmodellierung

- Sei $x_t, t = 1, 2, \dots$ die zeitdiskrete Zeitreihe von Interesse (Beispiel: Aktienkurs von Siemens)
- Sei $u_t, t = 1, 2, \dots$ eine weitere Zeitreihe, die Informationen über x_t liefert (Beispiel: Dow Jones)
- Der Einfachheit halber nehmen wir an, dass x_t und u_t skalar sind; Ziel ist die Vorhersage des nächsten Werts der Zeitreihe
- Wir nehmen ein System an der Form

$$x_t = f(x_{t-1}, \dots, x_{t-M_x}, u_{t-1}, \dots, u_{t-M_u}) + \epsilon_t$$

mit i.i.d. Zufallszahlen $\epsilon_t, t = 1, 2, \dots$. Diese modellieren unbekannte Störgrößen

Neuronale Netze in der Zeitreihenmodellierung (2)

- Wir modellieren durch ein Neuronales Netz

$$f(x_{t-1}, \dots, x_{t-M_x}, u_{t-1}, \dots, u_{t-M_u})$$

$$\approx f_{NN}(x_{t-1}, \dots, x_{t-M_x}, u_{t-1}, \dots, u_{t-M_u}, \mathbf{w}, \mathbf{v})$$

und erhalten als Kostenfunktion

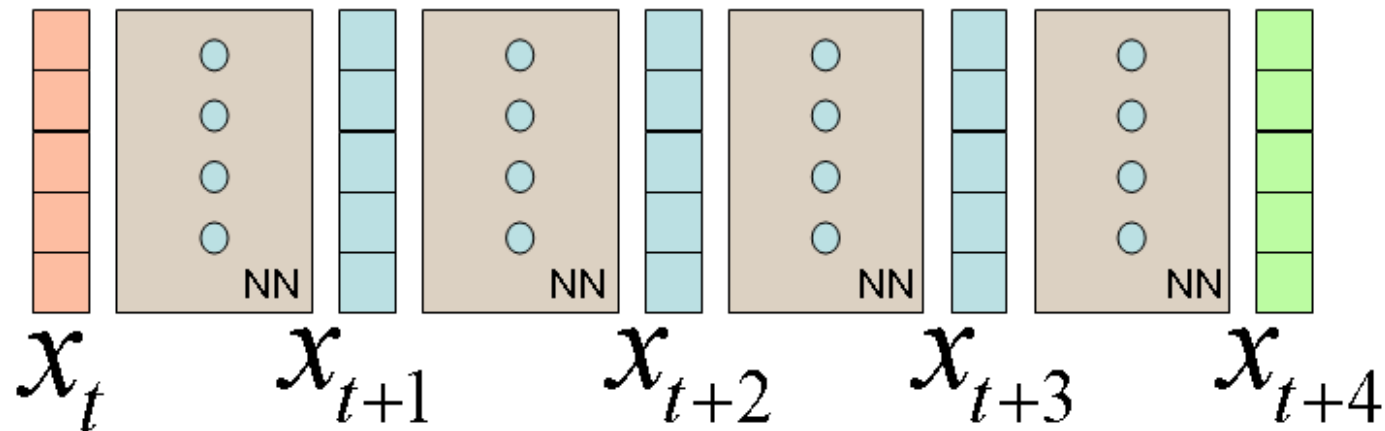
$$J_N(\mathbf{w}, \mathbf{v}) = \sum_{t=1}^N (x_t - f_{NN}(x_{t-1}, \dots, x_{t-M_x}, u_{t-1}, \dots, u_{t-M_u}, \mathbf{w}, \mathbf{v}))^2$$

- Das Neuronale Netz kann nun wie zuvor mit Backpropagation trainiert werden
- Das beschriebene Modell ist ein NARX Modell: **N**onlinear **A**uto **R**egressive Model with **e**xternal inputs. Andere Bezeichnung: TDNN (time-delay neural network)

Rekurrente Neuronale Netze

- Wenn x_t nicht direkt gemessen werden kann, sondern nur verrauschte Messungen zur Verfügung stehen (Verallgemeinerung des linearen Kalman Filters), reicht einfaches Backpropagation nicht mehr aus, sondern man muss komplexere Lernverfahren verwenden:
- Backpropagation through time (BPTT)
- Real-Time Recurrent Learning (RTRL)
- Dynamic Backpropagation

Illustration: BPPT



Vorwärtspropagation: basierend auf x_t wird x_{t+4} vorhergesagt;
Zielgrößen für x_{t+1} , ..., x_{t+3} sind unbekannt

Fehlerrückpropagation: basierend auf dem Fehler in der Prädiktion
bei x_{t+4} werden die Parameter adaptiert

Backpropagation-in-Time: Das Szenario entspricht
einem NN mit 7 versteckten Schichten und
Parameter-Sharing: Die Gewichte (Parameter) in allen
4 Netzen sind identisch

APPENDIX

Appendix: Approximationsgenauigkeit Neuronaler Netze

- Dies ist der entscheidende Punkt: *wieviele* innere Knoten sind notwendig, um eine vorgegebene Approximationsgenauigkeit zu erreichen?
- Die Anzahl der inneren Knoten, die benötigt werden, um eine vorgegebene Approximationsgenauigkeit zu erreichen, hängt von der Komplexität der zu approximierenden Funktion ab.
- Barron führt für das Maß der Komplexität der zu approximierenden Funktion $f(x)$ die Größe C_f ein, welche definiert ist als

$$\int_{\mathbb{R}^d} |w| |\tilde{f}(w)| dw = C_f,$$

wobei $\tilde{f}(w)$ die Fouriertransformation von $f(x)$ ist. C_f bestraft im Besonderen hohe Frequenzanteile.

- Die Aufgabe des Neuronalen Netzes ist es, eine Funktion $f(x)$ mit Komplexitätsmaß C_f zu approximieren.

- Der Eingangsvektors x ist aus \mathbb{R}^d , das Neuronale Netz besitzt n innere Knoten und die Netzapproximation bezeichnen wir mit $f_n(x)$
- Wir definieren als Approximationsfehler AF den mittleren quadratischen Fehler zwischen $f(x)$ und $f_n(x)$

$$AF = \int_{B_r} (f(x) - f_n(x))^2 \mu(dx). \quad (1)$$

μ ist ein beliebiges Wahrscheinlichkeitsmaß auf der Kugel $B_r = \{x : |x| \leq r\}$ mit Radius $r > 0$

- Barron hat nun gezeigt, daß für jede Funktion, für welche C_f endlich ist und für jedes $n \geq 1$ ein Neuronales Netz mit einer inneren Schicht existiert, so daß für den nach letzten Gleichung definierten Approximationsfehler AF_{Neur} gilt

$$AF_{Neur} \leq \frac{(2rC_f)^2}{n}. \quad (2)$$

- Dieses überraschende Resultat zeigt, daß die Anzahl der inneren Knoten *unabhängig* von der Dimension d des Eingangsraumes ist.

- Man beachte, daß für konventionelle Approximatoren mit festen Basisfunktionen der Approximationsfehler AF_{kon} exponentiell mit der Dimension ansteigt :

$$AF_{kon} \propto (1/n)^{2/d}.$$

- Für gewisse Funktionenklassen kann C_f exponentiell schnell mit der Dimension anwachsen, und es wäre somit nicht viel gewonnen. Barron zeigt jedoch, daß C_f für große Klassen von Funktionen nur recht langsam mit der Dimension (z.B. proportional) wächst.
- Quellen: Tresp, V. (1995). Die besonderen Eigenschaften Neuronaler Netze bei der Approximation von Funktionen. *Künstliche Intelligenz*, Nr. 4.

A. Barron. Universal Approximation Bounds for Superpositions of a Sigmoidal Function. *IEEE Trans. Information Theory*, Vol. 39, Nr. 3, Seite 930-945, Mai 1993.