Neuronale Verfahren zur Funktionsapproximation

- **Klassifikation mit dem Perceptron von Rosenblatt**
- **Vom Perceptron zum Multilagen-Perceptron**
- **Error-Backpropagation Lernregel**
- **Radiale Basisfunktionen-Netze**

Das Rosenblatt-Perceptron (1962)

Vorher: Regression. Jetzt: Binäre Klassifikation

- **Neuronale Struktur:**
  Das binäre Modell-Neuron

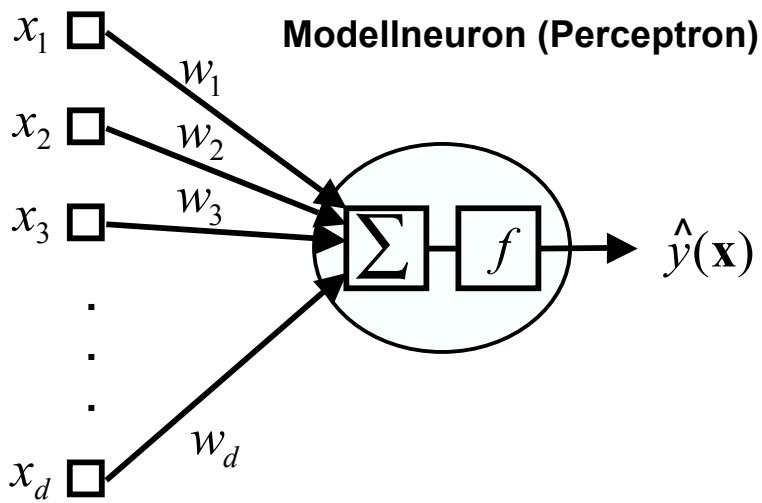
  $$\hat{y}(\mathbf{x}) = \Theta(\mathbf{w}^T\mathbf{x} + b), \quad y = \hat{y} + n$$

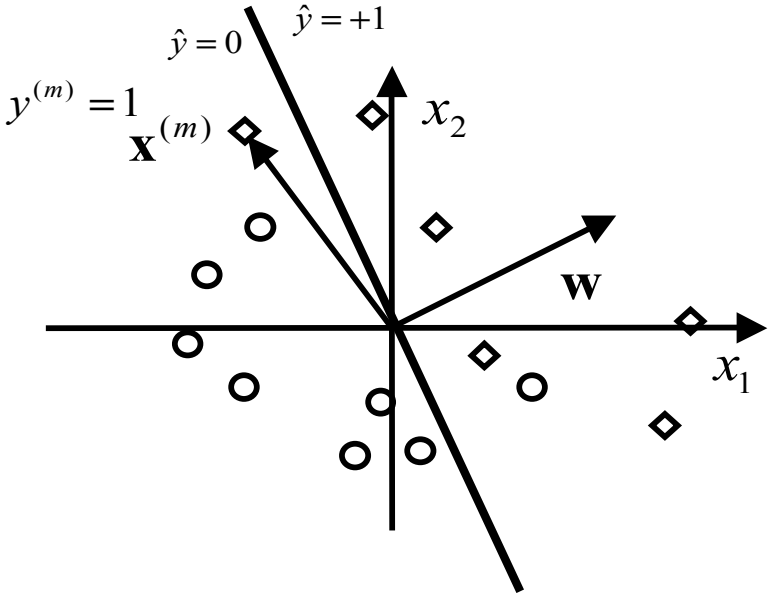
- **Wiederholung:**
  Das Perceptron kann als binärer Klassifikator dienen
  -- Geg: Daten in zwei Klassen, $y = 1, 0$
  -- Ges: Klassifikationsregel als Fkt. von $\mathbf{x}$
  -- Def: $x_{d+1} = 1,\ w_{d+1} = b \Rightarrow \hat{y} = \Theta(\mathbf{w}^T\mathbf{x})$

- **Lernen: Minimieren einer Fehlerfunktion. Hier: Gewichteter Falsch-Klassifikationsfehler**

- **Korrekte Klassifikation: Kein Fehler**
  Falsch-Klassifikation: positiver Fehler

$$F_P(w) = -\sum_{m=1}^{M} (y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\mathbf{w}^T\mathbf{x}^{(m)}$$





- **<u>Perceptron-Lernregel:</u> Gradientenabstieg**

$$\Delta\mathbf{w} = -\eta\nabla_{\mathbf{w}} F_P(\mathbf{w}) = \eta\sum_{m=1}^{M}(y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\,\mathbf{x}^{(m)} \quad \textbf{(Batch-Modus)}$$

Online-Lernregel:

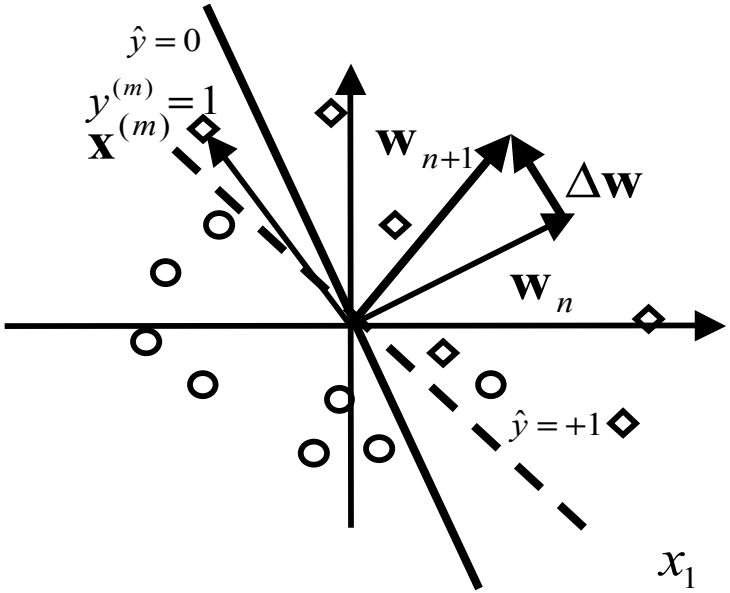
1. Wähle beliebiges $\mathbf{w}_0$

2. Wähle Muster m

3: $\mathbf{w}_{n+1} = \mathbf{w}_n + \eta(y^{(m)} - \hat{y}(\mathbf{x}^{(m)}))\,\mathbf{x}^{(m)}$

(d.h. tue nichts bei korrekter Klassifikation, biege $\mathbf{w}$ **bei Falsch-Klassifikation)**

4: Bis beste Klassifikation gehe zu 2



- **<u>Bem:</u>**

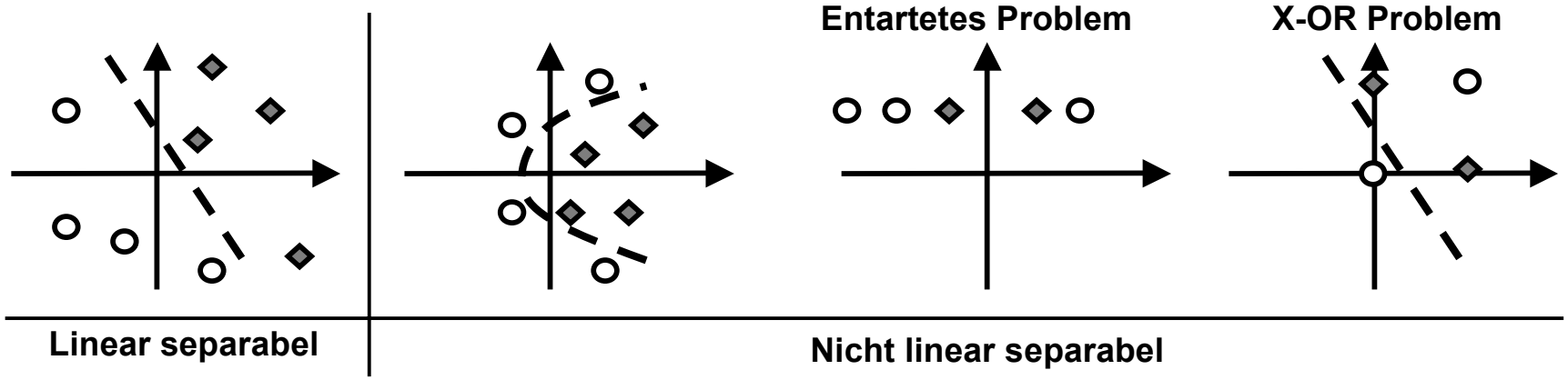
  -- Für linear separable Probleme <u>Konvergenzgarantie</u> in endl. vielen Schritten

  -- Funktioniert nur für linear separable Probleme

  -- Erweiterbar auf kontinuierliche Outputs

Lineare Separabilität:

Klassen können durch Hyperebene vollständig getrennt werden



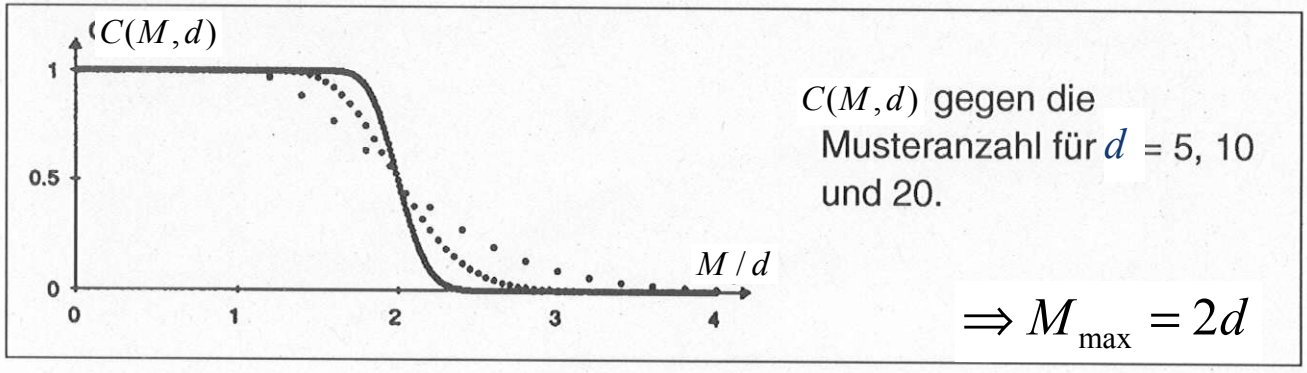
- **Wie wahrscheinlich sind M Muster in d Dimensionen linear separabel?**

  -- Wähle zufällig M d-dimensionale Muster

  -- Verteile zufällig Klassenlabels

  -- Bestimme Wa. für lin. Separabilität

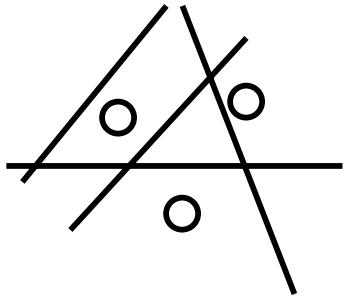
  -- Erg. f. hohe Dimensionen:



Behandlung nicht linear separabler Probleme 1: Dimensionalität

Lineare Klassifikation entspricht einer bestimmten Modellkomplexität
-- Wie komplex sind die Funktionen, die ein solches Modell implementieren kann?
--> „Zerschmettern“ von M Datenpunkten: Fähigkeit, alle 2^M möglichen Funktionen zu implementieren.

Def: Vapnik-Chervonenkis-Dimension (VC-Dimension) eines Modells:
Größtes M, das das Modell zerschmettern kann

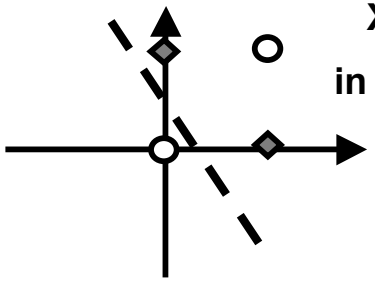


VC-Dimension e. linearen Klassifikators in d Dimensionen: $d+1$

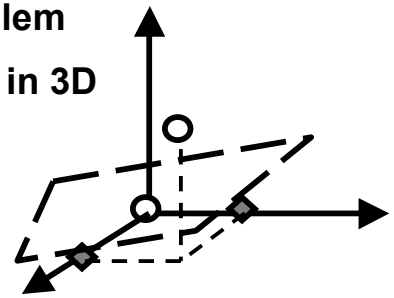
Lineare Klassifikation in höherer Dimension ... oder ...

X-OR Problem

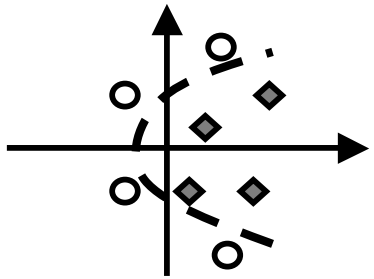
in 2D...



in 3D



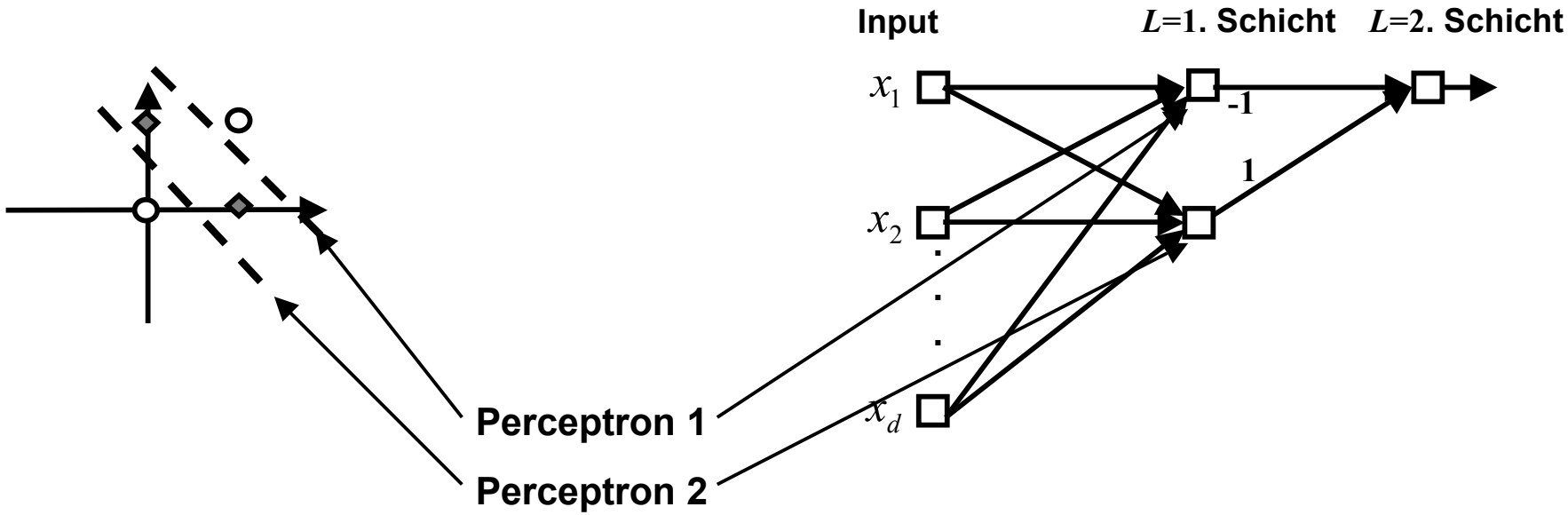
Nichtlineare Klassifikation



Vgl: Support Vector Machine (siehe später)

Vom Perceptron zum Multilagen-Perceptron

Behandlung nicht linear separabler Probleme 2: Mehrere Lagen



Perceptron:

- **Regression „komponentenweise nichtlinearer Funktionen“**
- **Klassifikation linear sparabler Probleme**

Multilagen Perceptron:

- **Allgemeine Funktionsapproximation**
- **Regression und Klassifikation**

Das Multilagen-Perceptron (MLP) für Regression

- **Motivation:**
  - **Nervenzellen im Gehirn sind hintereinandergeschaltet**
  - **Geschachtelte nichtlineare Transformationen sind mächtiger als eine einzige**

- **Aufbau**
  - **Feed-forward**
  - **Jedes Neuron gibt nichtlineare Funktion** g **des summierten Inputs weiter**

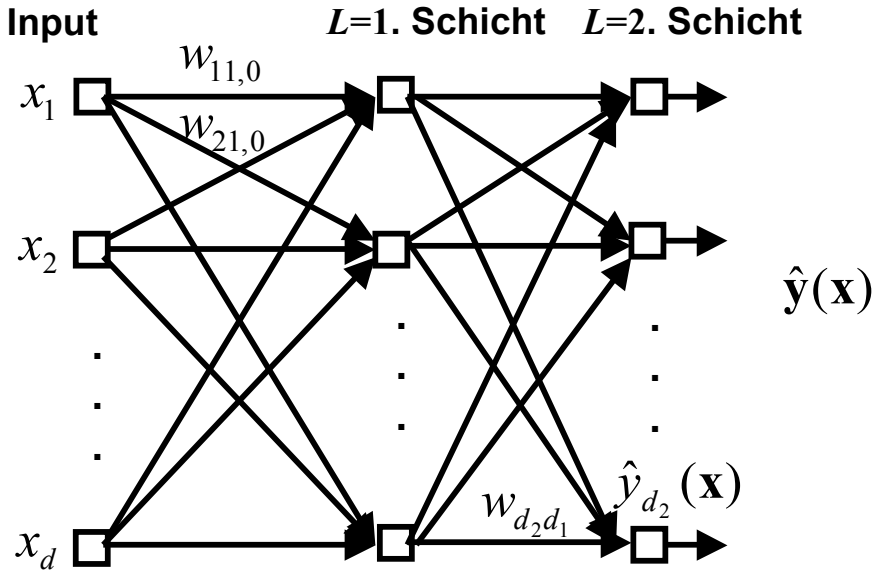
$$y_{i,0} = x_i, \quad d_0 = d$$

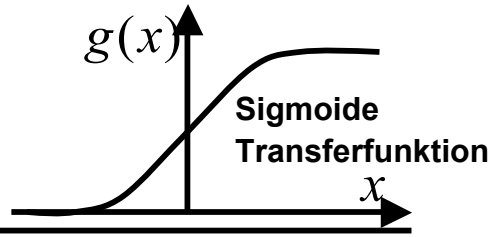
$$y_{i,L+1} = g\left(\sum_{j=1}^{d_L} w_{ij,L} y_{j,L} - \theta_i\right)$$

  - **mit** $d_L \to d_L + 1, \; y_{d_L+1,L} = -1, \; w_{id_L+1,L} = \theta_i$

$$y_{i,L+1} = g\left(\sum_{j=1}^{d_L} w_{ij,L} y_{j,L}\right)$$

  - **Bsp: zwei Schichten:** $\hat{y}_p \equiv y_{p,L=2} = g\left(\sum_{q=1}^{d_1} u_{pq} g\left(\sum_{r=1}^{d} v_{qr} x_r\right)\right)$





- **<u>Als Regressionsmodell geschrieben</u>**

$$y_i = \hat{y}_i(\mathbf{x}) + n_i = f_i(\mathbf{x} \mid \mathbf{w}) + n_i \equiv g\left(\sum_{j=1}^{d_L} w_{ij,L}\, g\left(\sum_{k=1}^{d_{L-1}} w_{jk,L-1} \cdots g\left(\sum_{q=1}^{d} w_{pq,0} x_q\right)\right)\right) + n_i$$

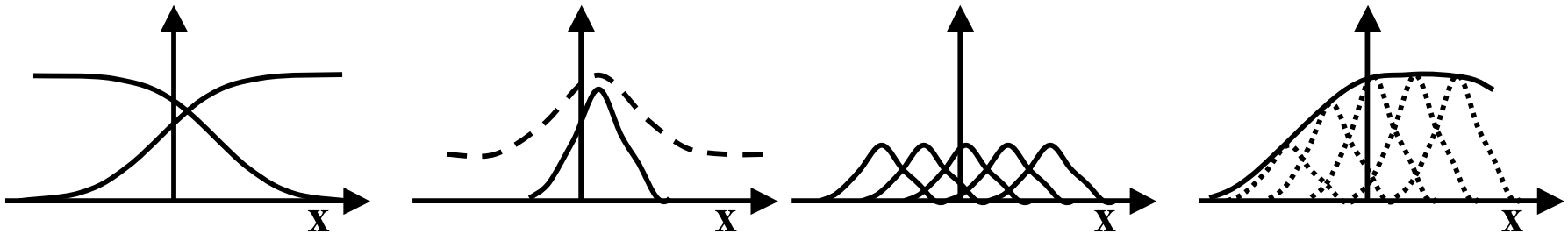
- **<u>Wichtigkeit der Nichtlinearität:</u> Falls g linear, entspricht MLP einschichtigem Fall**

$$y_i = \sum_{j=1}^{d_1} w_{ij,1} \sum_{k=1}^{d} w_{jk,0} x_k = \sum_{k=1}^{d} \left(\sum_{j=1}^{d_1} w_{ij,1} w_{jk,0}\right) x_k = \sum_{k=1}^{d} u_{ik} x_k$$

- **<u>Universale Approximationseigenschaft</u>**

  Ein dreischichtiges Netz kann jede beliebige kontinuierliche Funktion approximieren

  Anschaulich:

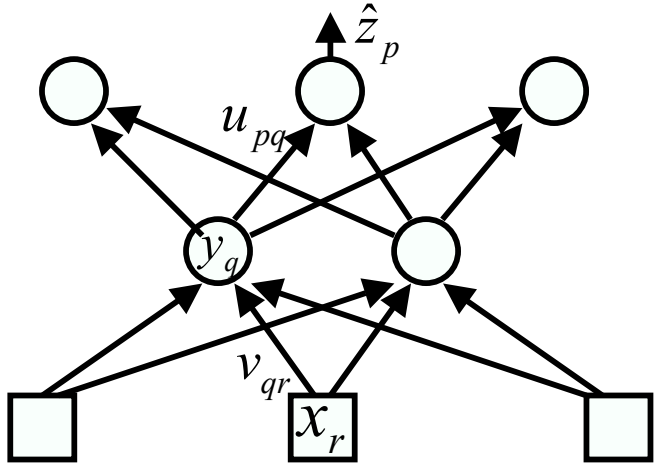


1. Schicht
Pool von kontinuierlichen Perceptrons

2. Schicht
Lokalisierte Antwort durch Kombination von Peceptron-Outputs plus sigmoide Transformation

3. Schicht
Kombination zur gewünschten Funktion

- **<u>Unklar</u>**
  - **Wieviele versteckte Neuronen werden benötigt**
  - **Wie sichert man sich gegen Überfitten ab? (=> Kreuzvalidierung)**
  - **Wie trainiert man die versteckten Neuronen?**
    => Error-Backpropagation-Algorithmus (Rumelhart, Werbos, `80er Jahre)



- **<u>Jetzt:</u> Betrachte zweischichtiges Netzwerk**

$$\hat{z}_p = g\left(\sum_q u_{pq} y_q\right) = g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r\right)\right)$$

- **Datensatz** $D = \{\mathbf{x}^{(m)}, \mathbf{z}^{(m)}\}, \; m = 1, \ldots, M$
- **Fehlerfunktion:**

$$F(\mathbf{w}) = \sum_m \sum_p (z_p^{(m)} - \hat{z}_p^{(m)})^2 = \sum_m \sum_p \left(z_p^{(m)} - g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r^{(m)}\right)\right)\right)^2, \quad \mathbf{w} \equiv (\mathbf{u}, \mathbf{v})$$

- **Modular:** $F(\mathbf{w}) = \sum_m E^{(m)}(\mathbf{w}) \qquad E^{(m)}(\mathbf{w}) = \sum_p \left(z_p^{(m)} - g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r^{(m)}\right)\right)\right)^2$

- **<u>Backpropagation Algorithmus: Herleitung</u>**

  --> Geschickte Anwendung des Gradientenabstiegs $\Delta\mathbf{w} = -\eta \nabla_{\mathbf{w}} E(\mathbf{w})$

  - **Def: Inputs** $a_p = \sum_q u_{pq} y_q, \quad b_q = \sum_r v_{qr} x_r$
  - **Def. Fehler:** $E(\mathbf{w}) = \sum_p \left(z_p - g\left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r \right) \right) \right)^2 =$

    (ohne index *m*)

$$= \sum_p \left(z_p - g\left(\sum_q u_{pq} g(b_q) \right) \right)^2 = \sum_p \left(z_p - g(a_p) \right)^2 = \sum_p \left(z_p - \hat{z}_p \right)^2$$

  - **Partielle Ableitungen für Versteckt-zu-Output Gewichte**

$$-\frac{\partial E}{\partial u_{ij}} = 2(z_i - \hat{z}_i)\frac{\partial \hat{z}_i}{\partial u_{ij}} = 2\underbrace{(z_i - \hat{z}_i) g'(a_i)}_{\delta_i} \frac{\partial a_i}{\partial u_{ij}} = \delta_i y_j$$

$$\boxed{\delta_i = 2(z_i - \hat{z}_i) g'(a_i)}$$

„Error“

  - **Partielle Ableitungen für Input-zu-Versteckt Gewichte**

$$-\frac{\partial E}{\partial v_{jk}} = \sum_p \underbrace{2(z_p - \hat{z}_p) g'(a_p)}_{\delta_p} \frac{\partial a_p}{\partial v_{jk}} = \sum_p \delta_p \frac{\partial}{\partial v_{jk}} \left(\sum_q u_{pq} g\left(\sum_r v_{qr} x_r \right) \right)$$

$$= \underbrace{\sum_p \delta_p u_{pj} g'(b_j)}_{=:\delta_j} \underbrace{\frac{\partial b_j}{\partial v_{jk}}}_{x_k} = \delta_j x_k,$$

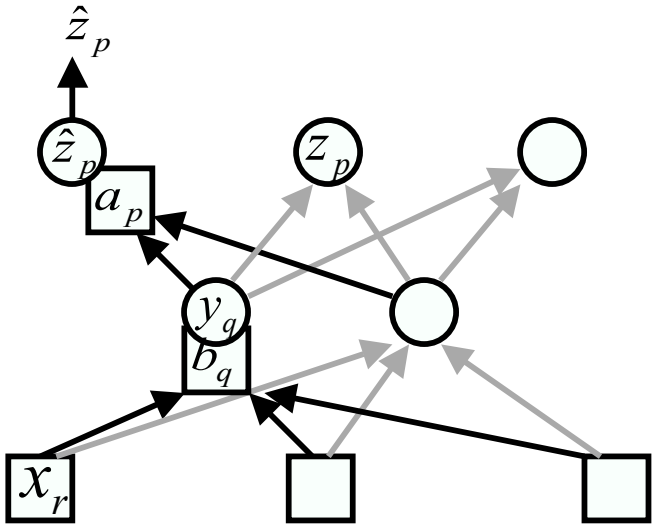
$$\boxed{\delta_j = \sum_p \delta_p u_{pj} g'(b_j)}$$

„Error-Backpropagation“

- **<u>Backprop-Algorithmus:</u>**
  Nach Zufallsbelegung der Gewichte auf kleine Werte
  - **Präsentiere Muster** $\mathbf{x}^{(m)}$

  <u>Signal forward-propagation:</u>
  - **Berechne und speichere von jeder Schicht die Inputs und Aktivitäten, also**

$$b_q = \sum_r v_{qr} x_r^{(m)}, \quad y_q = g(b_q)$$
$$a_p = \sum_r u_{pq} y_q, \quad \hat{z}_p = g(a_p)$$



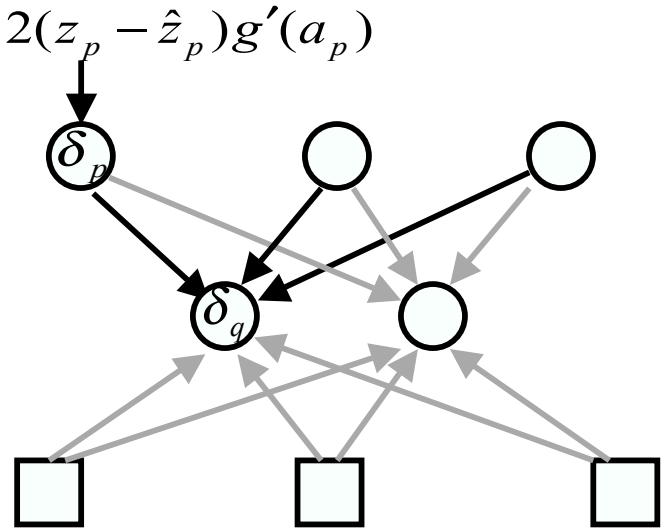
  - **Berechne Fehler an der Outputschicht**

$$\delta_p = 2(z_p - \hat{z}_p) g'(a_p)$$

  <u>Error-Backpropagation:</u>
  - **propagiere Fehler durch versteckte Schichten zurück**

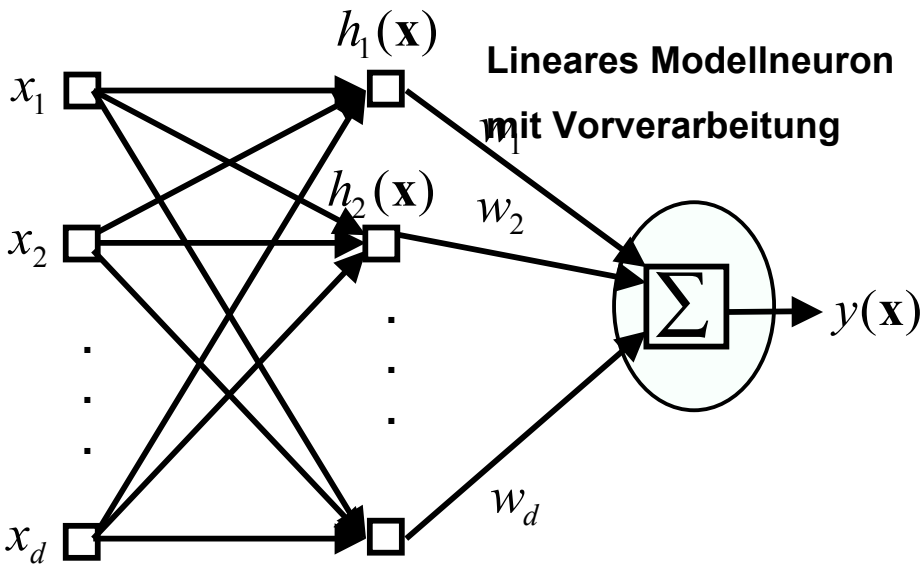
$$\delta_q = \sum_p \delta_p u_{pq} g'(b_q)$$

  - **Lerne mit Regel:** $\Delta u_{ij} = -\eta \dfrac{\partial E}{\partial u_{ij}} = \eta \delta_i y_j, \ \Delta v_{jk} = \eta \delta_j x_k$



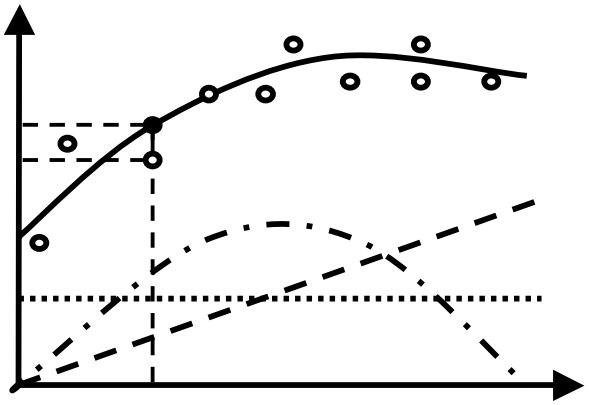
Wiederholung: Das lineare Modell

$$y(\mathbf{x}) = h_1(\mathbf{x})w_1 + h_2(\mathbf{x})w_2 + \ldots + h_d(\mathbf{x})w_d$$
$$= \sum_{i=1}^{d} h_i(\mathbf{x})w_i$$



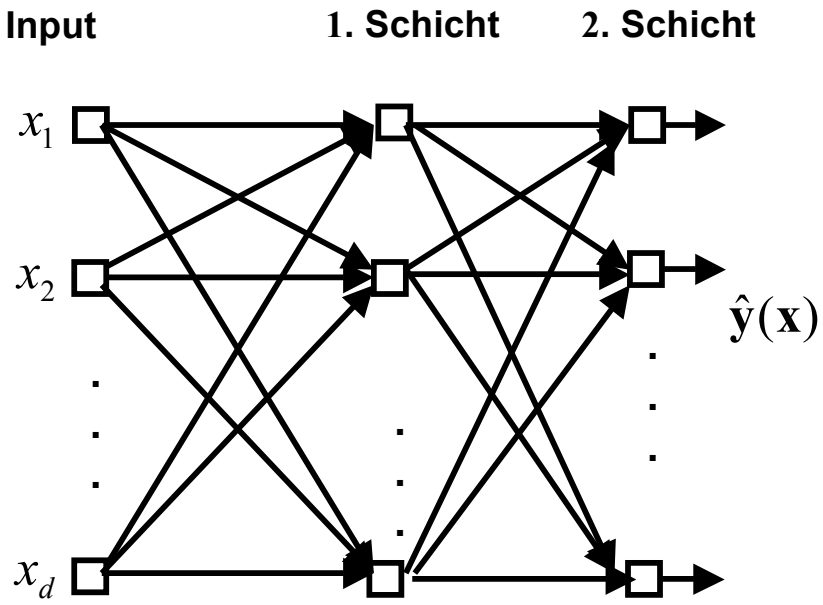
- **Funktionen $\mathbf{h}(\mathbf{x})$ sind von Hand vorgegeben, werden nicht gelernt**
- **Aber: Einfache Lösung**

analytisch berechenbar!



Radiale Basisfunktionen (RBF) Netzwerke

- **<u>Motivation:</u>**
  - **Hybrid zwischen linearem Modell und universalem Funktionsapproximator**
  - **Approximiere Output als lineare Summe nichtlinearer (Gauss-) Funktionen**
- **<u>Aufbau</u>**
  - **1. Schicht Gauss-Funktionen (anstatt $h(\mathbf{x})$) approximieren Segmente des Inputs**
  - **2. Schicht: Linearkombination der Gaussfunktionen**



  - **Bem: -- 1. Schicht nichtlinear, aber Gaussisch => effizient zu optimieren**
    -- 1. Schicht entspricht Schicht 1,2 des universalen Approximators
    -- 2. Schicht linear => einfache analytische ML Lösung

- **<u>Optimierung</u>**
  - **Optimiere μ,Σ der Gaussfunktionen nur basierend auf Input-Daten**
    -- Entspricht Mixture of Gaussian Dichteschätzer

  - **Verwende optimale Gaussfunktionen als Modellfunktionen eines linearen Modells**

Def: --

--