JOHANN WOLFGANG GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

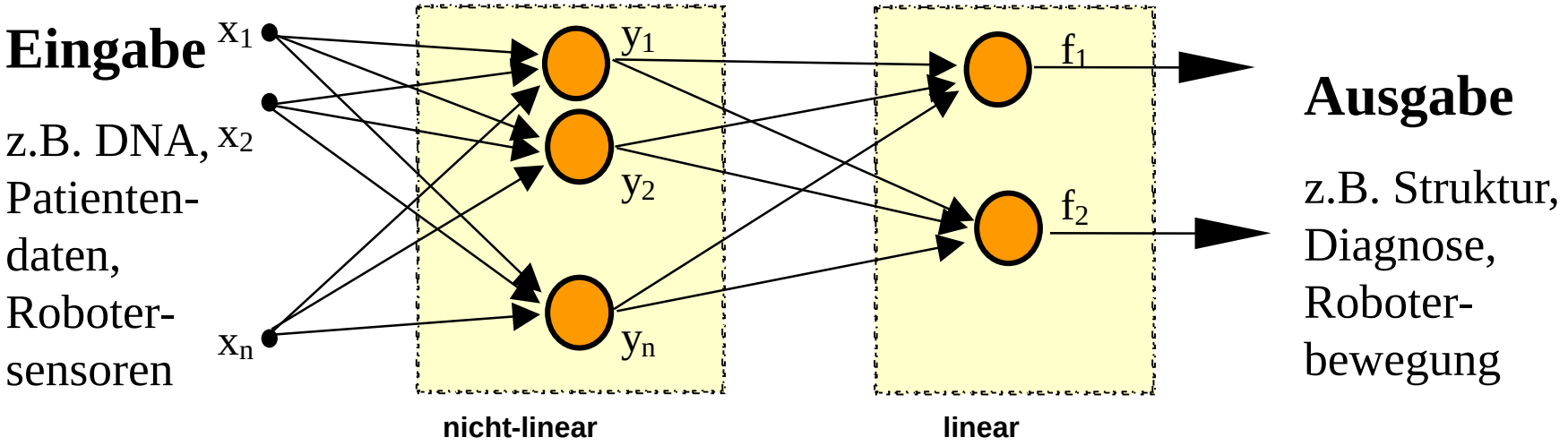
www.uni-frankfurt.de

Lernen mit RBF
(Radialen Basis-Funktionen)

Praktikum Adaptive Systeme

Mehrschichten-Netze

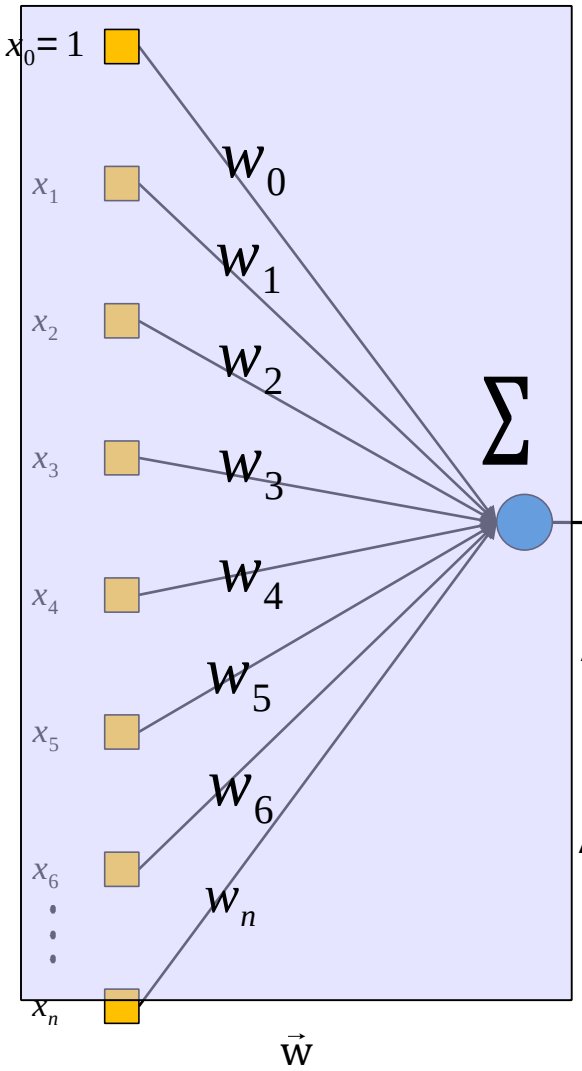
Fähigkeiten von Mehrschicht-Netzen:



- Ein 2-Schichtennetzwerk mit nicht-linearer Ausgabefunktion S(z) kann JEDE beliebige Funktion so genau wie gewünscht approximieren, wenn genügend Neuronen ex.

Aber: Neuronenzahl=?? Gewichte = ??

Wiederholung - Perzeptron



$$y = S(z)$$

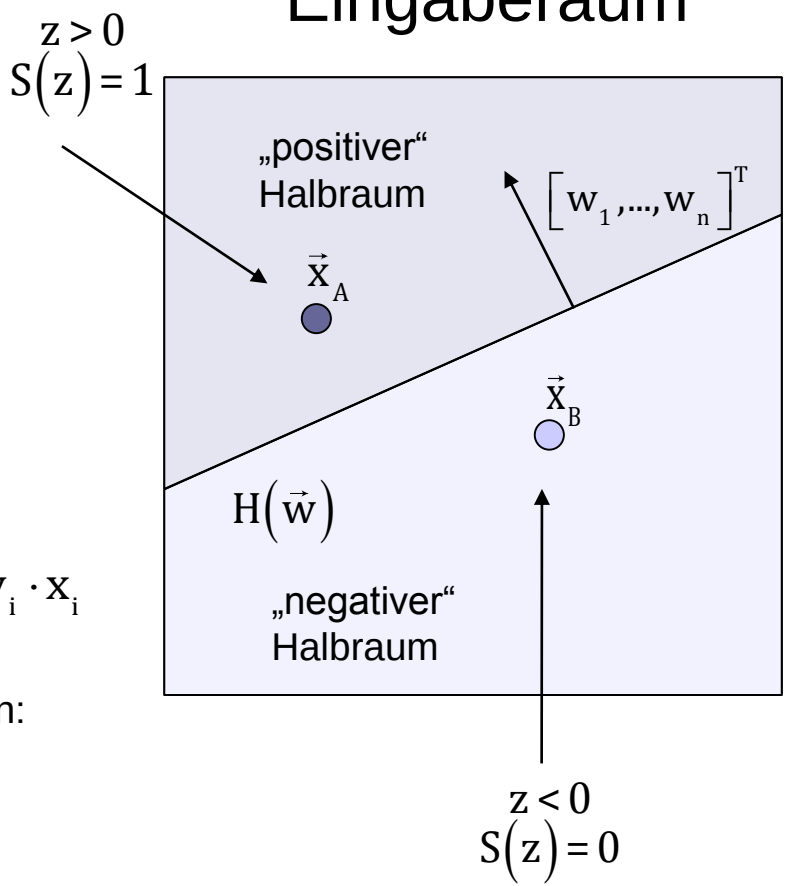
Aktivierung:

$$z = w_0 + \sum_{i=1}^{n} w_i \cdot x_i$$

Ausgabefunktion:

$$S(z) = \begin{cases} 1, & z > 0 \\ 0, & z \leq 0 \end{cases}$$

Klassifikation mit Perzeptron:



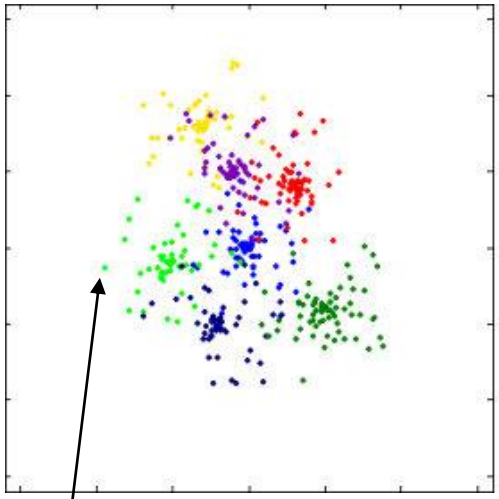
Klassifikation durch ein Perzeptron

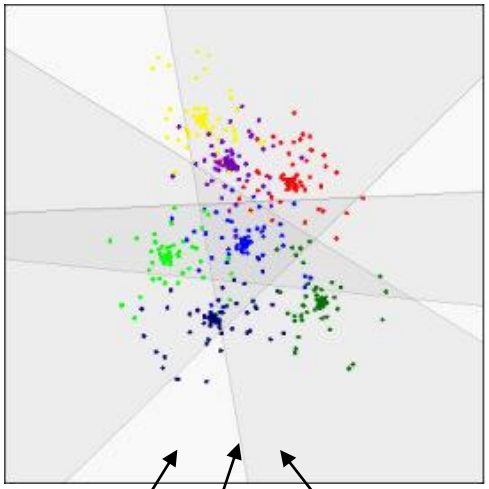
JOHANN WOLFGANG GOETHE
UNIVERSITÄT
FRANKFURT AM MAIN

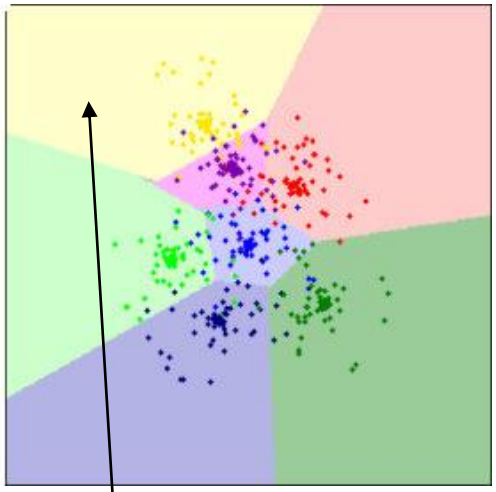
Muster 7 verschiedener Klassen im Eingaberaum

Trennung der Klassen durch **mehrere** Perzeptrons (dargestellt für blaue Menge)

Resultierende Entscheidungsgebiete







Muster (Klasse „hellgrün“)

Trennende Hyperebene eines Perzeptrons

negativer Halbraum

positiver Halbraum

Entscheidungsgebiet (Klasse „gelb“)

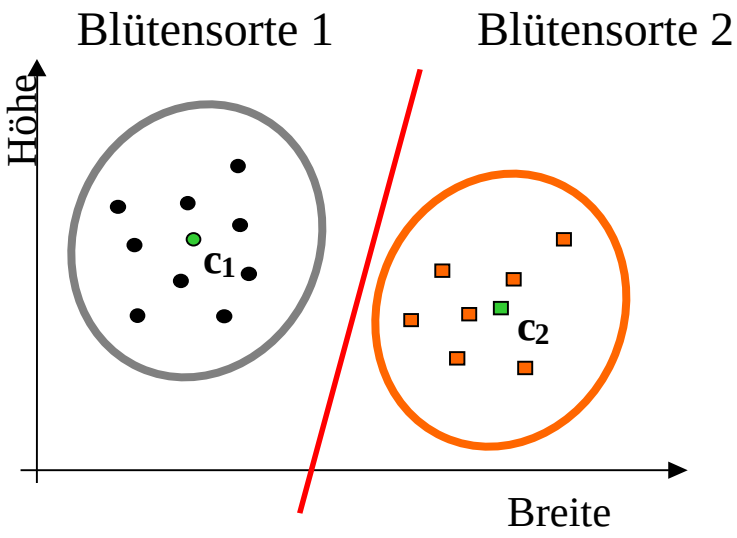
Problem „Inkrementelles Lernen“

Problem Perzeptron:

Ein neues Neuron wirkt sich bei **allen** Mustern aus.

Lösung:

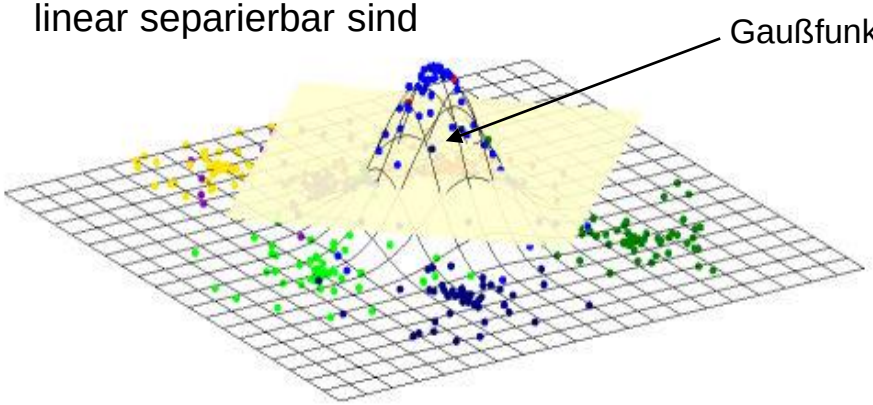
Im Musterraum **nur lokal** wirkende Neuronen

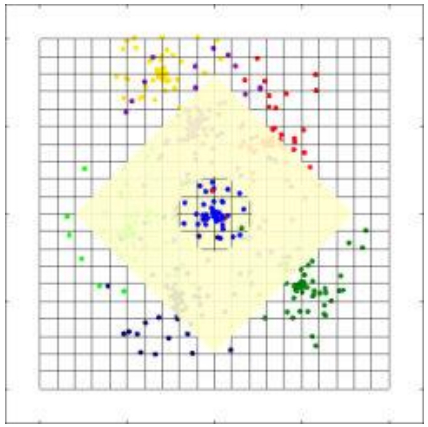


Klassifikation mit Radialen Basisfunktionen

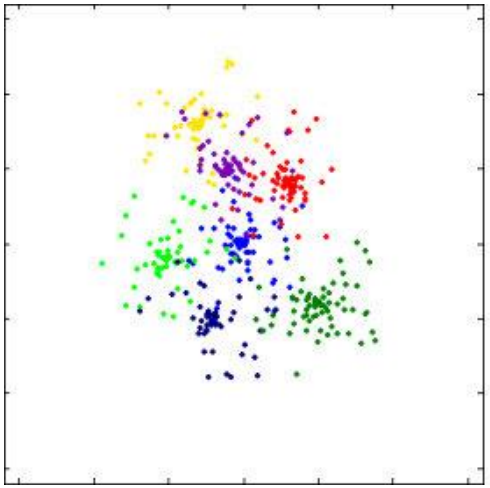
Bessere Klassifikation durch gekrümmte Entscheidungsgebiete

Idee: Nichtlineare Abbildung in neuen Raum, in dem die Muster (mit höherer Wahrscheinlichkeit) linear separierbar sind

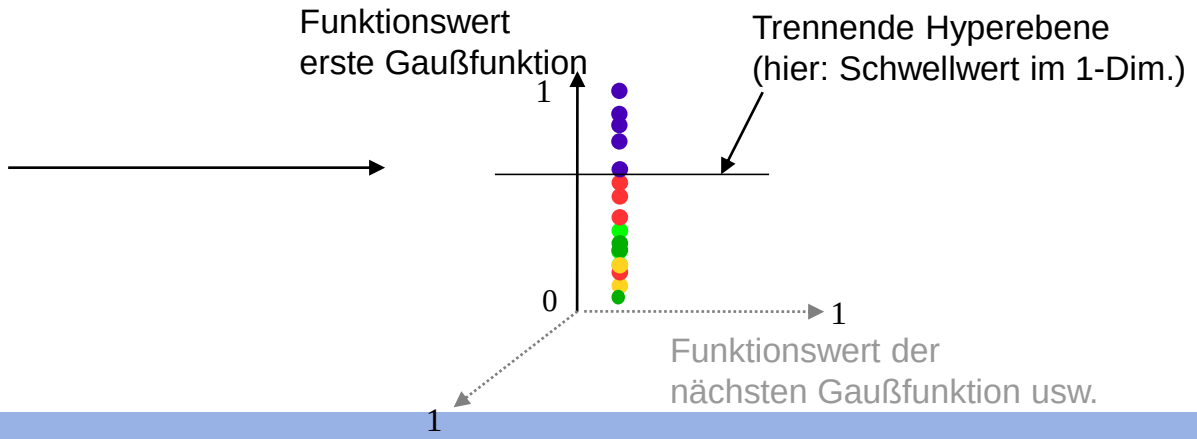




Eingaberaum (xy-Koordinaten)

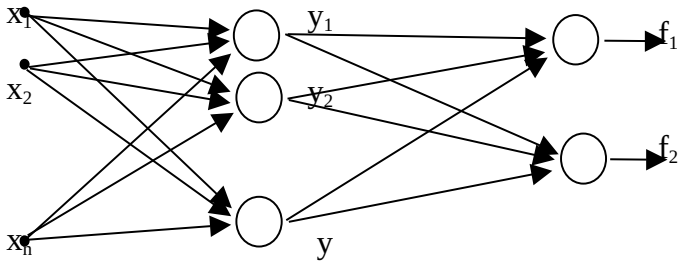


Neuer Raum (Punkte der Gauß-Funktionswerte)



RBF-Netze

Typisch: 2-Schichten Netzwerk

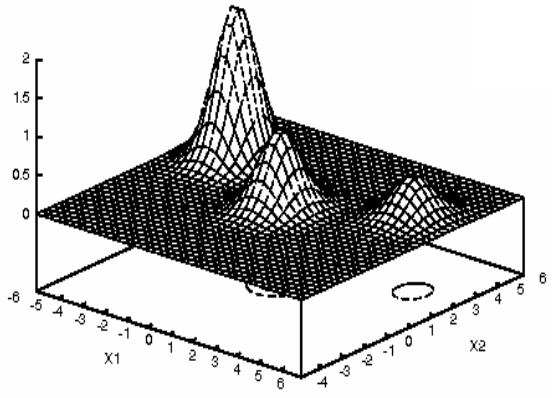


Aktivität

nicht normiert

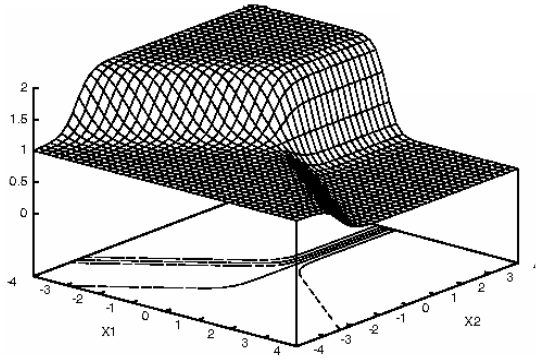
$$f_i(\mathbf{x}) = \sum_{k=1}^{m} w_k \, y_k = \sum_{k=1}^{m} w_k \, S_k(\mathbf{x})$$

$$\text{mit} \quad S(\mathbf{c}, \mathbf{x}) = e^{\frac{-(\mathbf{c}_k - \mathbf{x})^2}{2\sigma^2}}$$



normiert

$$f_i(\mathbf{x}) = \sum_{k=1}^{m} w_k y_k = \frac{\sum_{k=1}^{m} w_k S_k(\mathbf{x})}{\sum_{j=1}^{m} S_j(\mathbf{x})}$$

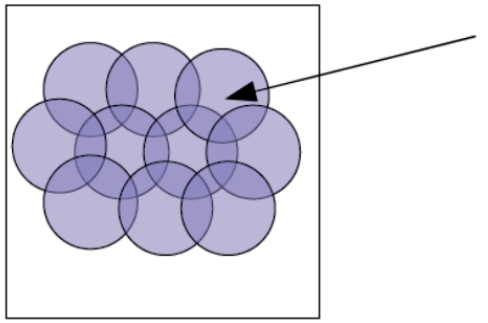


1. Schicht Lernen

- Gleichmäßige Rasterung des Eingaberaums
- Setzen von RBF-Zentren ausreichend breit auf die ersten *N* Eingabesamples.
- Zufälliges Setzen der Zentren, normierte Breite
  $\mathbf{c}_i = \mathbf{x}_i$, $\sigma_i = 1$, Aufhören nach *N* RBF
- Lernen der Zentren und Weiten durch Backpropagation
- Anzahl und Weiten initial vorgeben, Lernen durch Kohonen-Netz
- ...

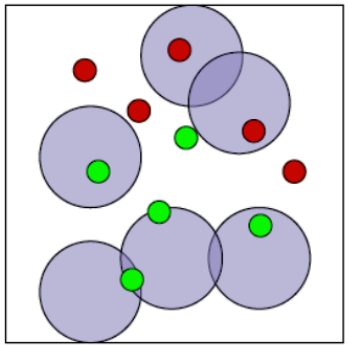
Wahl der RBF-Parameter

Wahl der RBF-Radien σ

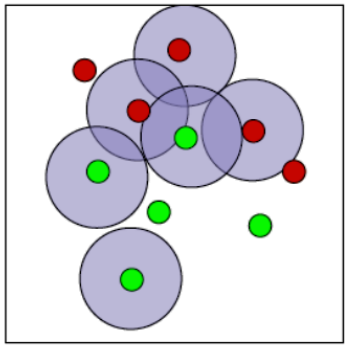


Wahl der RBF-Zentren (Gewichtsvektoren)

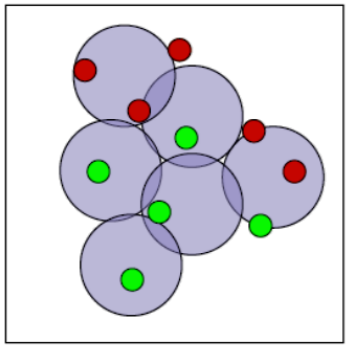
Zufällig im Eingaberaum verteilt



Random Sampling

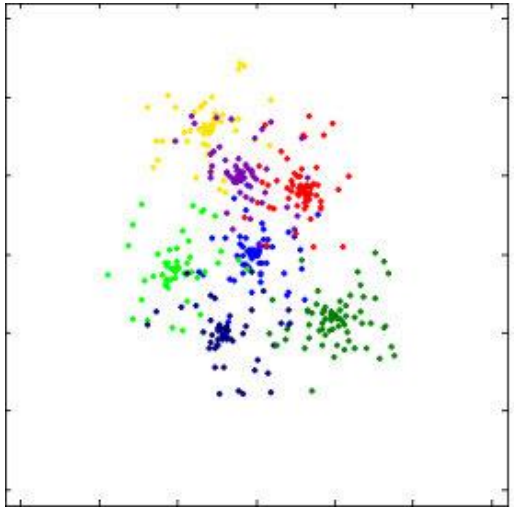


Wahl der RBF-Zentren durch
Kohonennetz oder Neuronengas

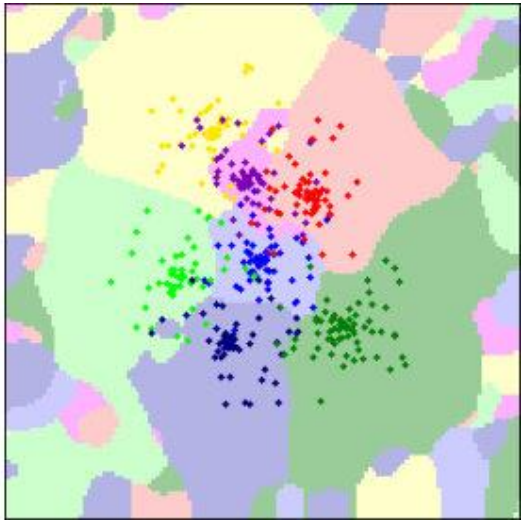


RBF-Ergebnisse

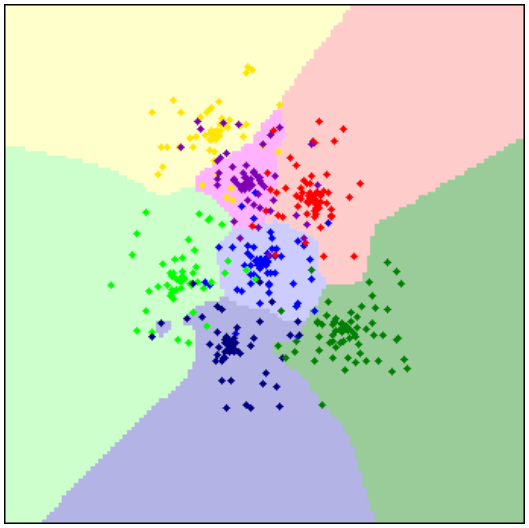
Eingaberaum



Zufällige Verteilung der RBF-Zentren im Eingaberaum



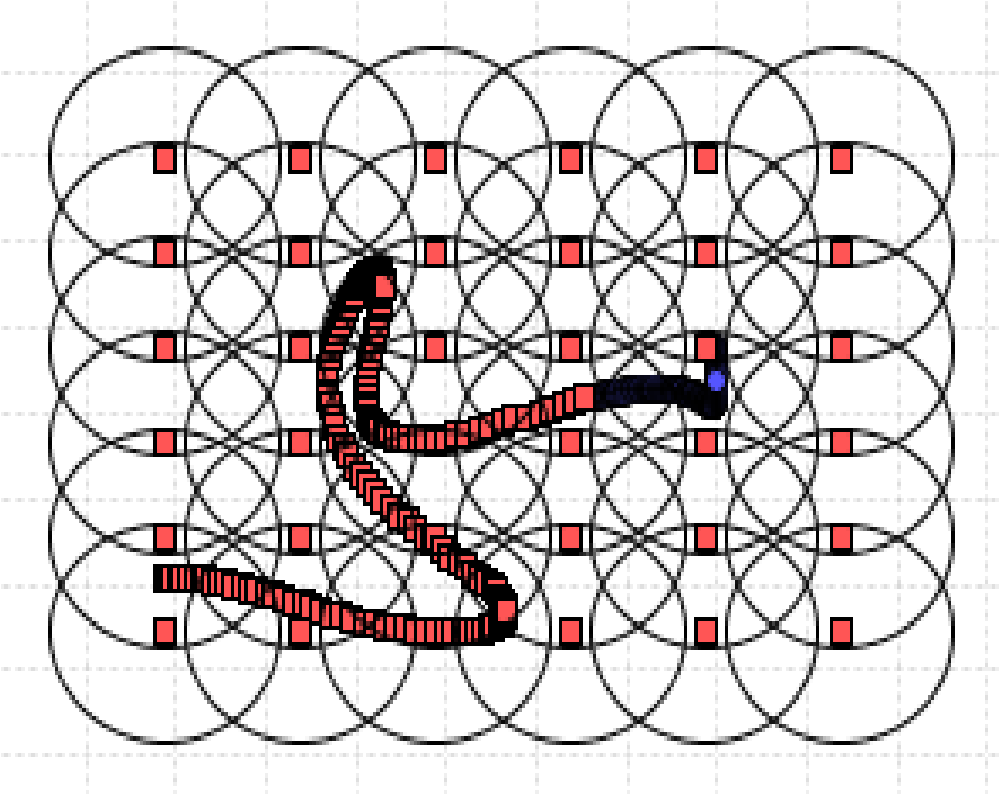
Wahl der RBF-Zentren durch Random-Sampling



RBF-Probleme

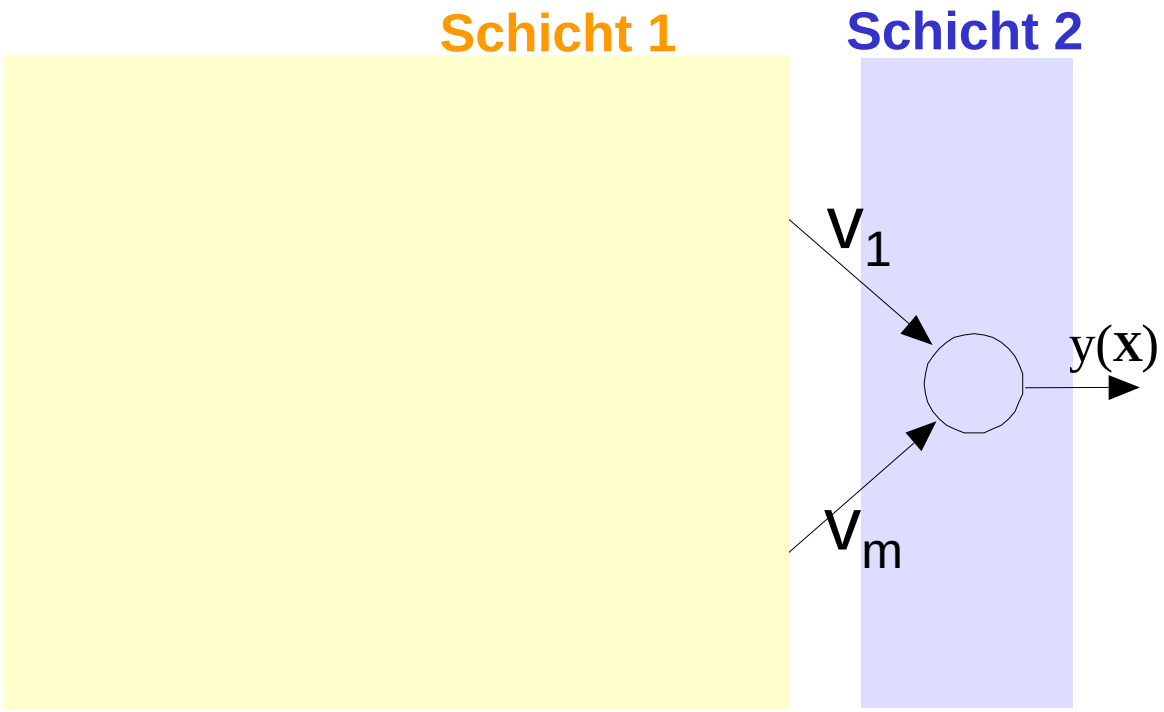
- Sigmoidale Ausgabefkt auch für **Extrapolation**,
- RBF-Ausgabefkt **nur** für **Intrapolation**.

Problem: Vorhersage durch *untrainierte* RBF-Neuronen



2.Schicht: Anpassung

Normiertes RBF-Netz



$$y(\mathbf{x}) = \hat{f}(\mathbf{x}) = \sum_i w_i v_i \qquad \text{mit } v_i = \tilde{S}_i(\mathbf{x}, \mathbf{c}_i)$$

$$\mathbf{w}(t) = \mathbf{w}(t-1) - \gamma(t)\,(\mathbf{w}^T\mathbf{v} - \hat{f}(\mathbf{x}))\,\mathbf{v}$$

Delta-Lernregel

RBF Code

```
γ:= 0.1;                                  (* Lernrate festlegen *)
REPEAT
    Read( PatternFile,x,L)                (* Eingabe *)
    (* Aktivität bilden im Netz *)
     Sum := 0.0;
     FOR i:=1 TO m DO                     (* Für alle Neuronen der 1. Schicht *)
          v[i]   : =   S_rbf(x–x0[i]);    (* Nicht-lin. RBF-Ausgabe*)
          Sum : =   Sum+v[i];             (* Gesamtaktivität bilden für Normierung*)
     END;
     f := Z(w,v);                         (* Aktivität 2.Schicht: f(x)=w^T v *)
     f := f/Sum;                          (* und normieren*)
    (* Lernen der Gewichte der 2.Schicht *)
     FOR i:=1 TO m DO                     (* Für alle Dimensionen *)
          w[i]:= w[i] – γ*(f–L)*v[i]/Sum  (* Gewichte verändern: delta-Regel *)
     END;
UNTIL EndOf(PatternFile)
```

Fragen ?