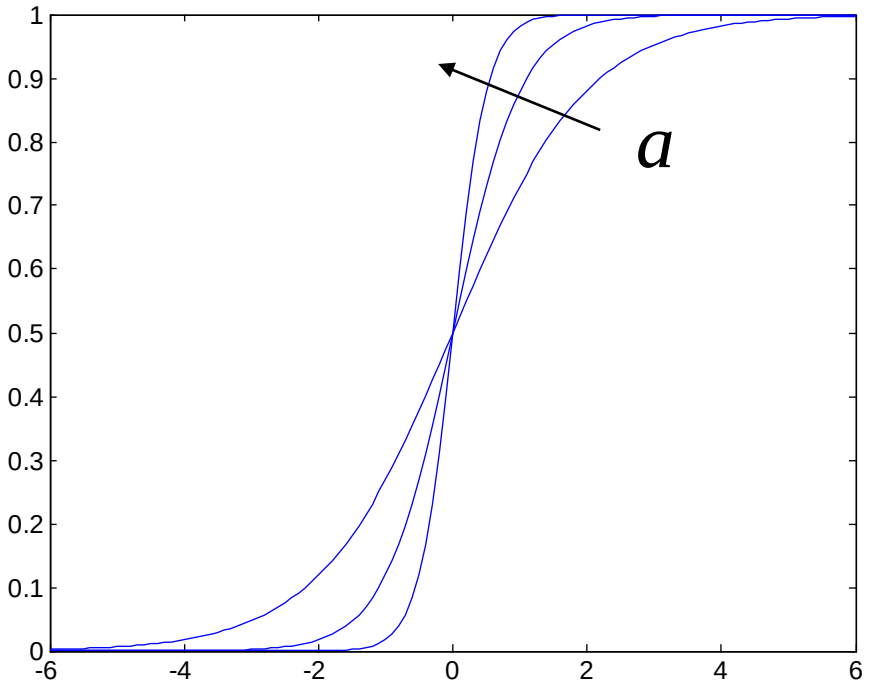
Lernstrategien für Neuronale Netze - Der Backpropagation-Algorithmus

Im Unterschied zu den bisher behandelten NNs mit Schwellwertfunktionen sind die NNe mit *stetig differenzierbaren nichtlinearen Aktivierungsfunktionen f(x).* Am weitesten verbreitet ist die *Sigmoid-Funktion* $\psi(x)$:

$$f(x) = \psi(x) = \frac{1}{1+e^{-ax}}$$

Sie approximiert mit wachsendem *a* die Sprungfunktion $\sigma(x)$.



Die Funktion $\psi(x)$ hat einen engen Bezug zur tanh-Funktion. Es gilt für a=1:

$$\tanh(x) = \frac{2}{1+e^{-2x}} - 1 = 2\psi(2x) - 1$$

Fügt man die zwei Parameter a und b hinzu, so erhält man eine etwas allgemeinere Form:

$$\psi_g(x) = \frac{1}{1+e^{-a(x-b)}} - 1$$

wobei mit a die Steilheit und mit b die Position auf der x-Achse beeinflusst werden kann.

Die Nichtlinearität der Aktivierungsfunktion ist entscheidend für die Existenz des Multi-Lagen-Perceptron; ohne diese, würde das mehrschichtige in ein triviales lineares einschichtiges Netzwerk zusammenschrumpfen.

Die Differenzierbarkeit von f(x) erlaubt die Anwendung von notwendigen Bedingungen ($\partial(\cdot)/\partial w_{i,j}$) zur Optimierung der Gewichtskoeffizienten $w_{\mathrm{i,j}}$.

Die erste Ableitung der Sigmoid-Funktion kann auf sich selbst zurückgeführt werden:

$$\frac{d\psi}{dx} = \frac{-1}{(1+e^{-x})^2}(-1)e^{-x} = \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = \psi(x)(1-\psi(x))$$

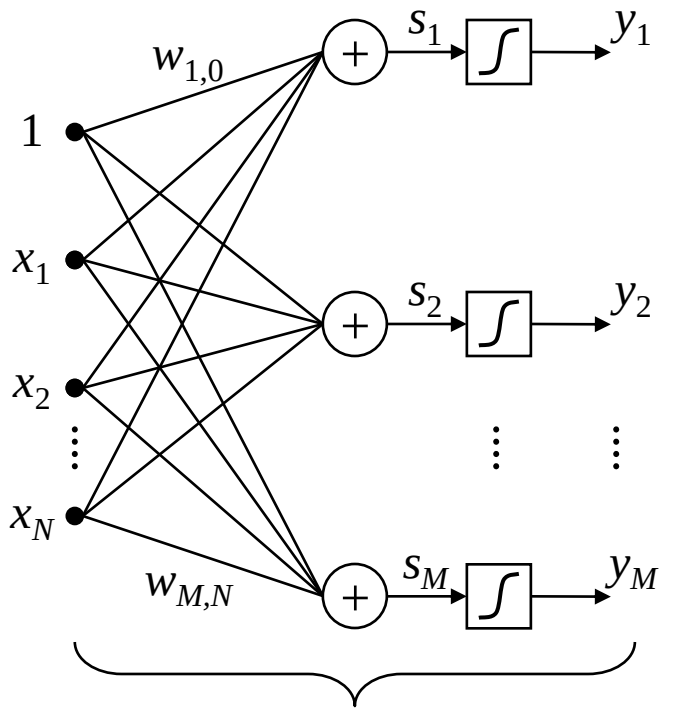
Eine Schicht des Neuronalen Netzes

Eine einzige Schicht des NN wird charakterisiert durch die $M \times N$ Koeffizienten der Gewichtsmatrix $\mathbf{W}$, den Offset-Vektor $\mathbf{b}$ und eine vektorielle Sigmoid-Funktion $\boldsymbol{\psi}$.
Nach einer Erweiterung des Eingangsvektors um eine konstante 1

$$\mathbf{x}' = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$

ergibt sich eine Schicht in der nebenstehenden Form.

Ihre Funktion lässt sich beschreiben durch eine Matrixmultiplikation, gefolgt von einer nichtlinearen Aktivierungsfunktion, welche in identischer Form auf alle Elemente angewandt wird.



$$\mathbf{y} = \boldsymbol{\psi}(\mathbf{W}\mathbf{x} + \mathbf{b}) = \boldsymbol{\psi}(\mathbf{W}'\mathbf{x}') = \boldsymbol{\psi}(\mathbf{s})$$

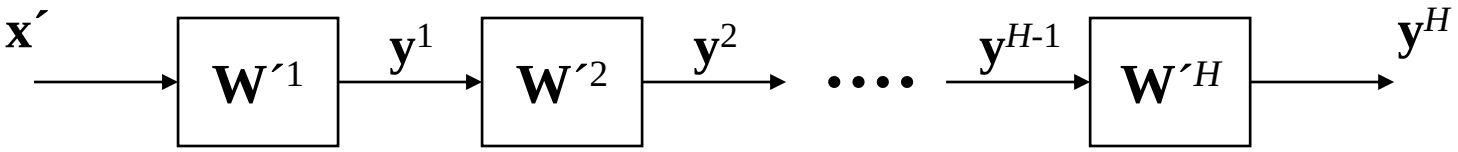
mit: $\mathbf{x}' = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$ und $\mathbf{W}' = [\mathbf{b}, \mathbf{W}]$

Gestalt der erweiterten Gewichtsmatrix

$$\mathbf{W}' = \begin{bmatrix} w_{1,0} = b_1 & w_{1,1} & w_{1,2} & \cdots & w_{1,N} \\ w_{2,0} = b_2 & w_{2,1} & w_{2,2} & \cdots & w_{1,N} \\ w_{3,0} = b_3 & w_{3,1} & w_{3,2} & \cdots & w_{1,N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{M,0} = b_M & w_{M,1} & w_{M,2} & \cdots & w_{M,N} \end{bmatrix} = \left[\mathbf{b}, \mathbf{W}\right]$$

Das Neuronale Netz mit *H* Schichten

Das mehrlagige NN mit *H* Schichten hat in jeder Schicht eine eigene Gewichtsmatrix $\mathbf{W}'^{i}$, jedoch identische Sigmoid-Funktionen:



$$\mathbf{y}^H = \boldsymbol{\psi}(\mathbf{W}'^H \boldsymbol{\psi}(\ldots \mathbf{W}'^2 \boldsymbol{\psi}(\mathbf{W}'^1 \mathbf{x}') \ldots))$$

Das Lernen basiert auf der Anpassung der Gewichtsmatrizen, mit dem Ziel der Minimierung eines Fehlerquadrat-Kriterums zwischen dem Sollwert $\mathbf{y}$ und der Approximation $\hat{\mathbf{y}}$ durch das Netz (überwachtes Lernen). Der Erwartungswert ist zu bilden über das zur Verfügung stehende Trainings-Ensemble von $\{\hat{\mathbf{y}}_j, \mathbf{x}_j\}$:

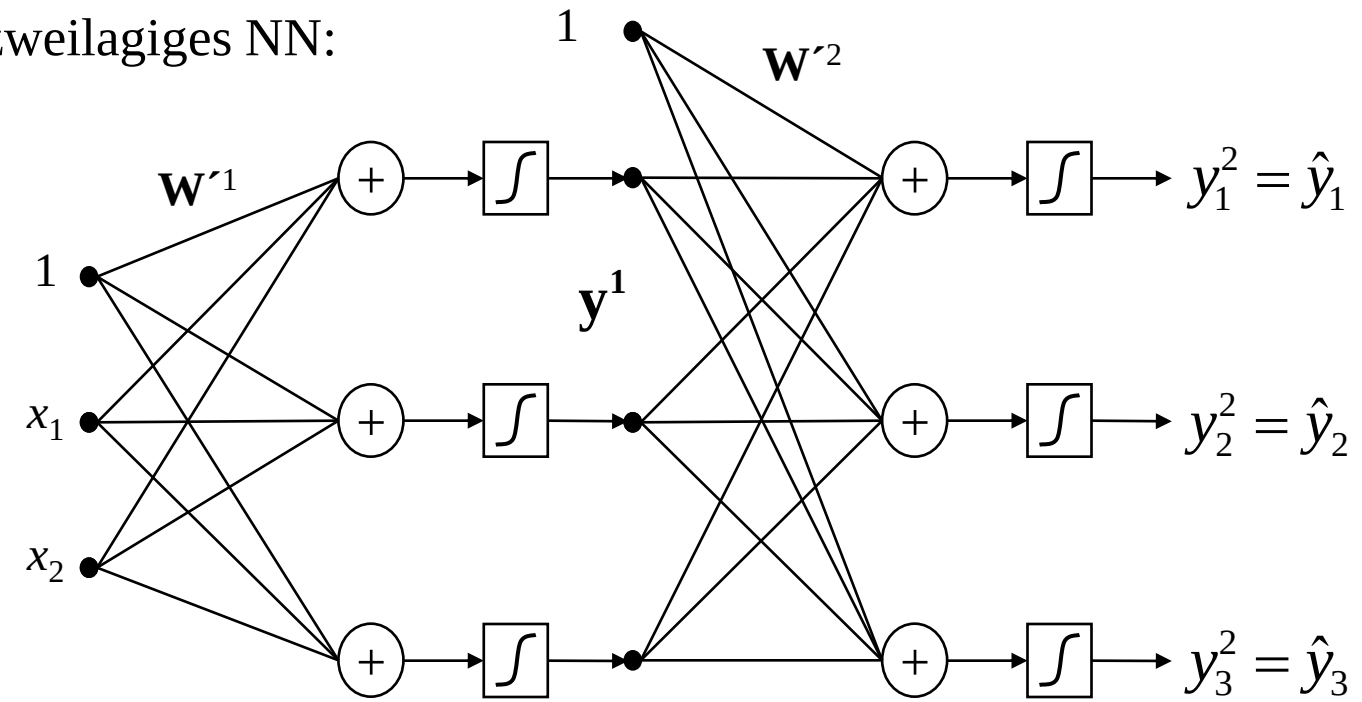
$$J = \min_{\mathbf{W}'^i} E\left\{\left\|\hat{\mathbf{y}} - \mathbf{y}\right\|^2\right\} = \min_{\mathbf{W}'^i} E\left\{\left\|\mathbf{y}^H - \mathbf{y}\right\|^2\right\}$$

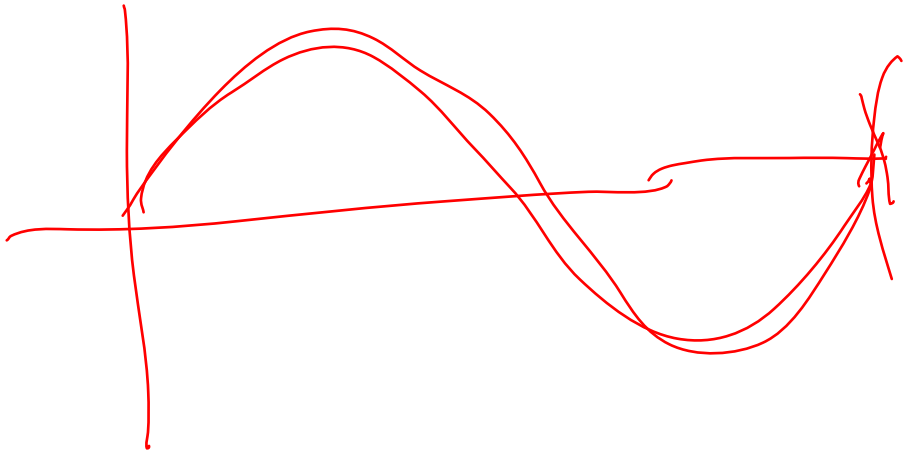
Anmerkung: Das einschichtige NN und der lineare Polynomklassifikator sind identisch, wenn die Aktivierungsfunktion $\psi \equiv 1$ gesetzt wird.

Beziehungen zu dem Konzept der Funktionsapproximation durch eine Linearkombination von Basisfunktionen

Bei der Approximation z.Bsp. mit Polynomfunktionen sind die Basisfunktionen von vornherein festgelegt. Bei dem NN werden die Basisfunktionen iterativ durch Parametermatrizen aufgebaut. Zudem ist die Linearkombination nicht der letzte Schritt der Approximation; vielmehr geht die Wirkung durch eine nichtlineare Abbildung hervor und der Vorgang wiederholt sich!

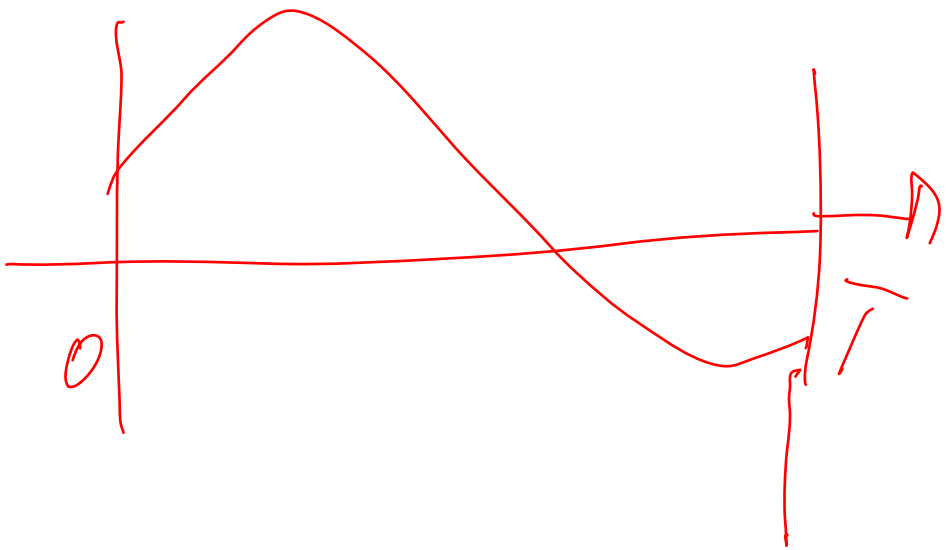
Beispiel für ein zweilagiges NN:





$$x(t) = \sum_i \alpha_i \cdot \varphi_i(t)$$

$$\| \hat{x}(t) - x(t) \|^2$$



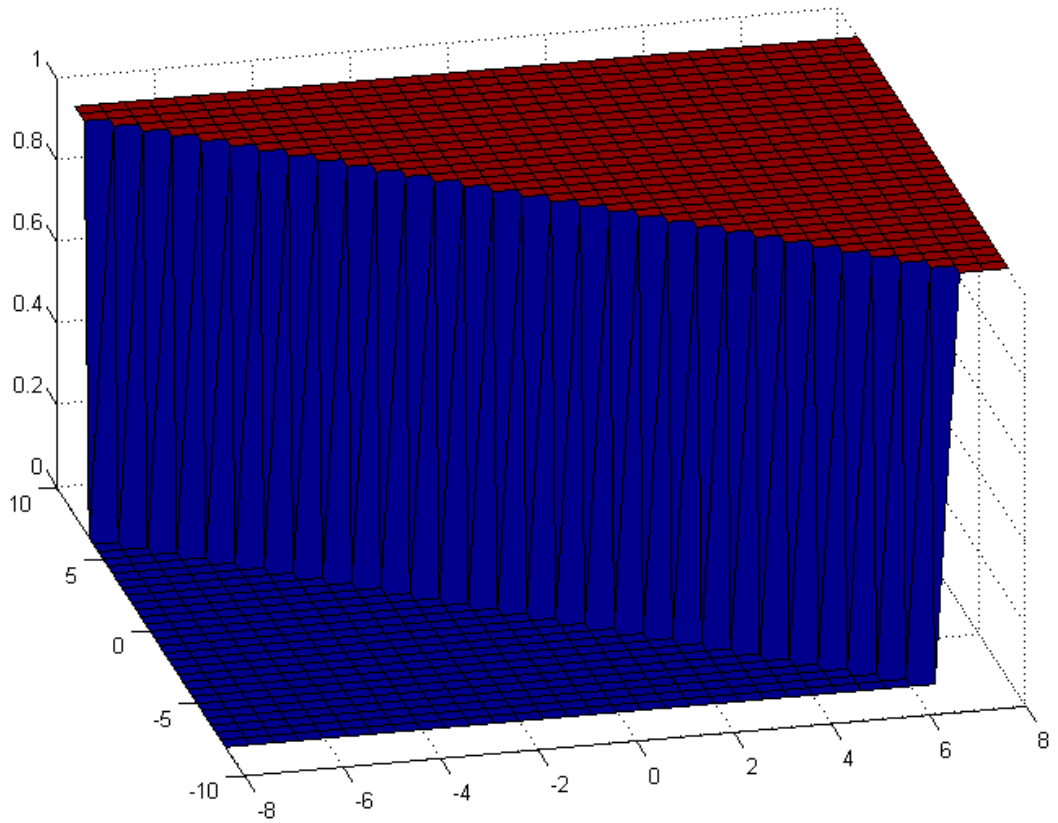
$$x(t) = \sum_{i=1}^{k} \alpha_i \cdot \sin(\omega_i \cdot t + \varphi_i)$$

Beziehungen zu dem Konzept der Funktionsapproximation durch eine Linearkombination von Basisfunktionen

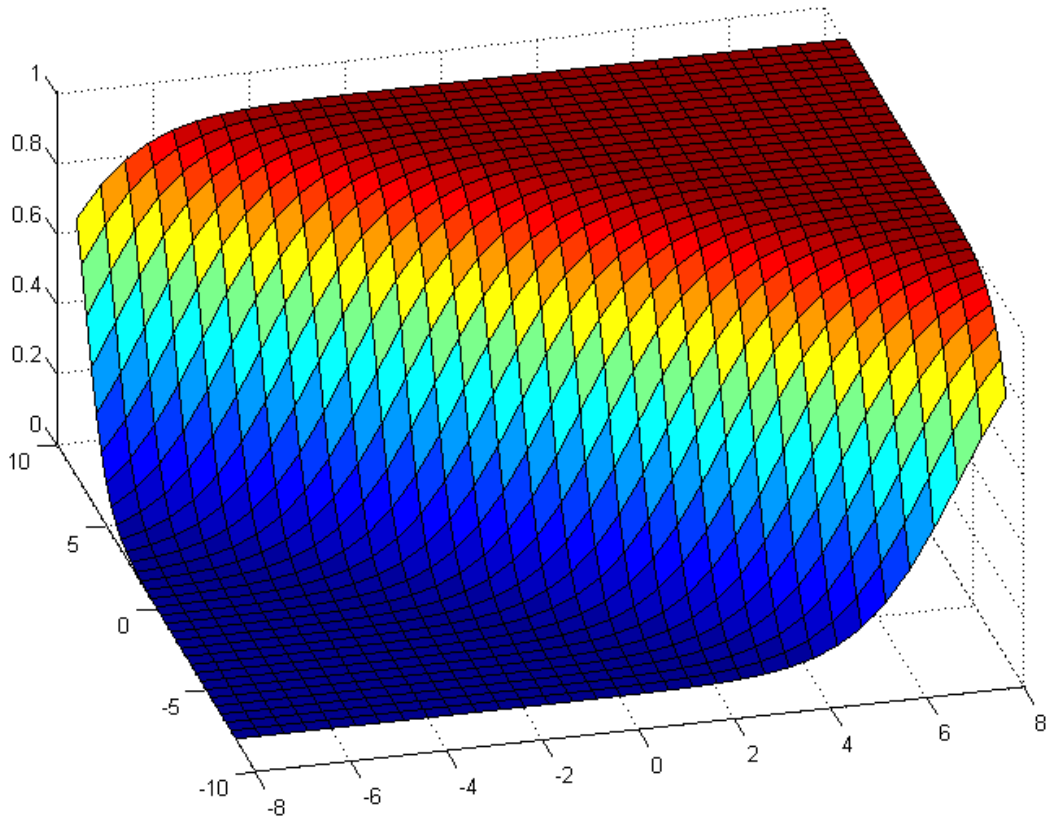
Die erste Schicht erzeugt die Basisfunktionen in Form des verdeckten Vektors $\mathbf{y}_1$ und die zweite Schicht bildet eine Linearkombination dieser Basisfunktionen.

Deshalb beeinflusst die Koeffizientenmatrix $\mathbf{W}'^1$ der ersten Schicht das *Aussehen* der Basisfunktionen, während die Gewichtmatrix $\mathbf{W}'^2$ der zweiten Schicht die *Koeffizienten der Linearkombinationen* enthält. Diese wird zusätzlich durch die Aktivierungsfunktion gewichtet.

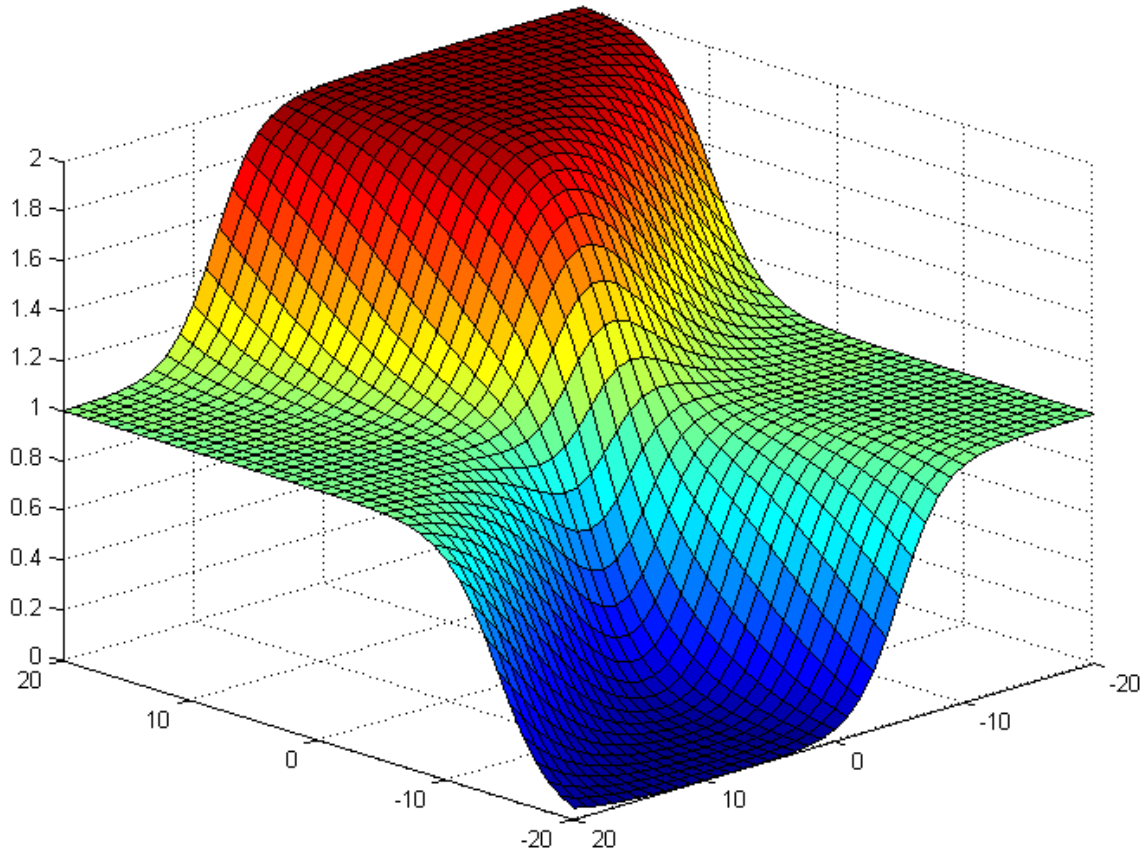
Reaktion eines Neurons mit zwei Eingängen und einer Schwellwertfunktion σ



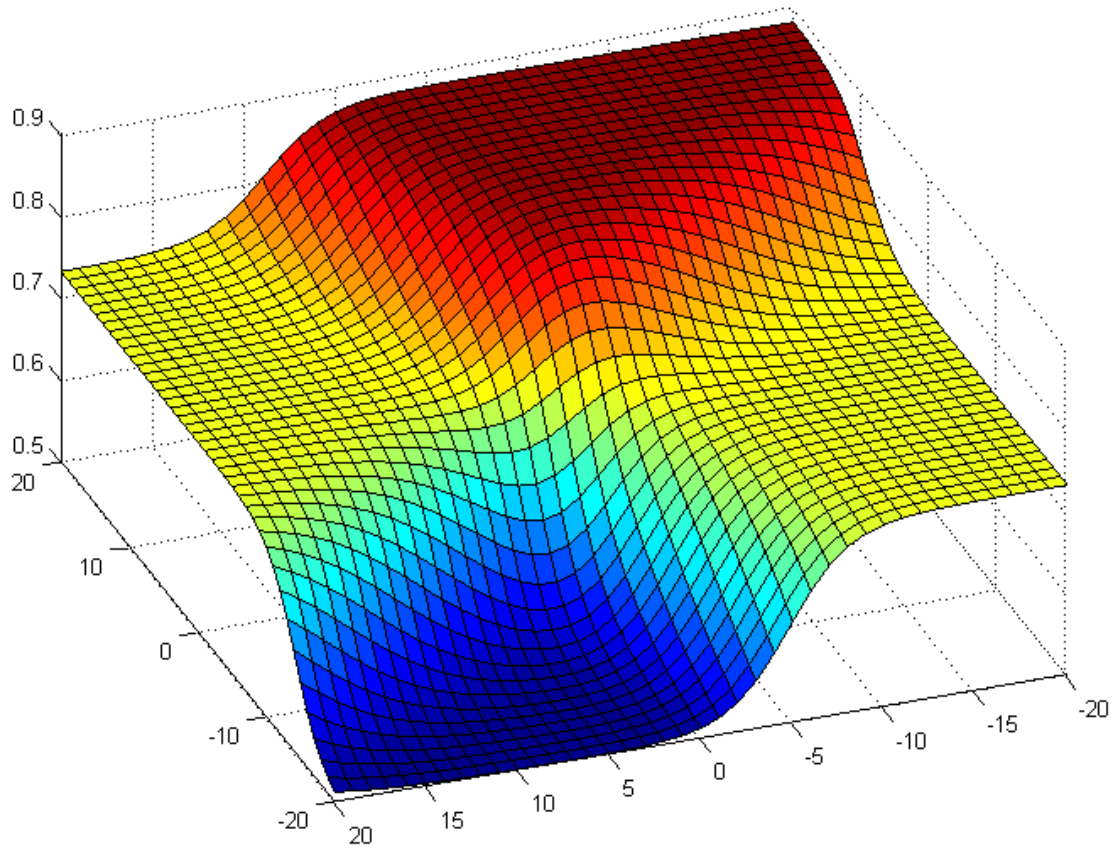
Reaktion eines Neurons mit zwei Eingängen und einer Sigmoidfunktion ψ



Überlagerung zweier Neuronen mit je zwei Eingängen und einer Sigmoidfunktion



Reaktion eines Neurons in der *zweiten* Schicht auf zwei Neuronen der ersten Schicht nach der Bewertung durch die Sigmoidfunktion



Der Backpropagation-Lernalgorithmus

- Die Klassenzugehörigkeitsabbildung geschieht durch ein Multilagenperceptron, dessen i-ter Ausgang eine 1 erzeugt in den Regionen von **x**, welche durch die Stichproben $\mathbf{x}_i$ der entsprechenden Bedeutungsklasse bevölkert ist, und sie erzeugt eine 0 in Gebieten, welche durch andere Bedeutungsklassen belegt sind. In den Gebieten dazwischen und außerhalb findet eine Inter- bzw. eine Extrapolation statt.
- Das Netzwerk wird trainiert auf der Grundlage der Optimierung des quadratischen Gütekriteriums:

$$J = E\left\{\left\|\hat{\mathbf{y}}(\mathbf{W}'^{h}, \mathbf{x}') - \mathbf{y}\right\|^2\right\}$$

- Dieser Ausdruck ist nichtlinear sowohl in den Elementen des Eingangsvektors **x´**, als auch in den Gewichtskoeffizienten $\{w'_{nm}{}^{h}\}$.

- Nach Einsatz der Funktionsabbildung des Multilagenperceptrons erhält man:

$$J = E\left\{\left\| \underbrace{\boldsymbol{\psi}(\mathbf{W}'^{H} \ldots \boldsymbol{\psi}(\mathbf{W}'^{2}\boldsymbol{\psi}(\mathbf{W}'^{1}\mathbf{x}')\ldots))}_{\hat{\mathbf{y}}} - \mathbf{y} \right\|^2 \right\}$$

- Gesucht ist das *globale* Minimum. Es ist jedoch nicht klar, ob ein solches existiert oder wie viele *lokale* Minima vorhanden sind.
  Der Backpropagation Algorithmus löst das Problem iterativ mit einem Gradientenalgorithmus. Iteriert wird gemäß:

$$\boxed{\mathbf{W}' \leftarrow \mathbf{W}' - \alpha \nabla \mathbf{J}} \quad \text{mit: } \mathbf{W}' = \left\{\mathbf{W}'^{1}, \mathbf{W}'^{2}, \ldots, \mathbf{W}'^{H}\right\}$$

$$\text{und d. Gradienten: } \nabla \mathbf{J} = \frac{\partial J}{\partial \mathbf{W}'} = \left\{\frac{\partial J}{\partial w'^{h}_{nm}}\right\}$$

- Die Iteration wird beendet, wenn der Gradient bei einem (lokalen oder auch globalen) Minimum verschwindet.
- Eine geeignete Wahl von α ist schwierig. Kleine Werte erhöhen die Anzahl der Iterationen. Größere Werte vermindern zwar die Wahrscheinlichkeit in ein lokales Minimum zu laufen, aber man riskiert, daß das Verfahren divergiert und das Minimum nicht gefunden wird (oder Oszillationen auftauchen).

- Die Iteration hat leider nur eine *lineare* Konvergenzordnung (Gradientenalgorithmus).
- Berechnung des Gradienten:
  Zur Bestimmung der zusammengesetzten Funktion

$$\hat{\mathbf{y}}(\mathbf{x}') = \boldsymbol{\psi}(\mathbf{W}'^{H} \ldots \boldsymbol{\psi}(\mathbf{W}'^{2}\boldsymbol{\psi}(\mathbf{W}'^{1}\mathbf{x}')\ldots))$$

- wird die Kettenregel benötigt.
- Der Fehler, verursacht durch eine Stichprobe ergibt sich zu:

$$J_j = \tfrac{1}{2}\left\|\hat{\mathbf{y}}(\mathbf{x}_j) - \mathbf{y}_j\right\|^2 = \tfrac{1}{2}\sum_{k=1}^{N^H}(\hat{y}_k(j) - y_k(j))^2$$

- Der Erwartungswert $E\{...\}$ des Gradienten muss durch den Mittelwert über alle Stichprobengradienten approximiert werden:

$$\nabla\mathbf{J} = \tfrac{1}{n}\sum_{j=1}^{n}\nabla\mathbf{J}_j$$

- Man unterscheidet zwischen
  - individuellem Lernen aufbauend auf die letzte Stichprobe

$$\left[\mathbf{x}_j, \mathbf{y}_j\right] \Rightarrow \nabla \mathbf{J}_j$$

  - und dem kumulativen Lernen (batch learnung), aufbauend auf alle Stichproben

$$\nabla \mathbf{J} = \frac{1}{n} \sum_{j=1}^{n} \nabla \mathbf{J}_j$$

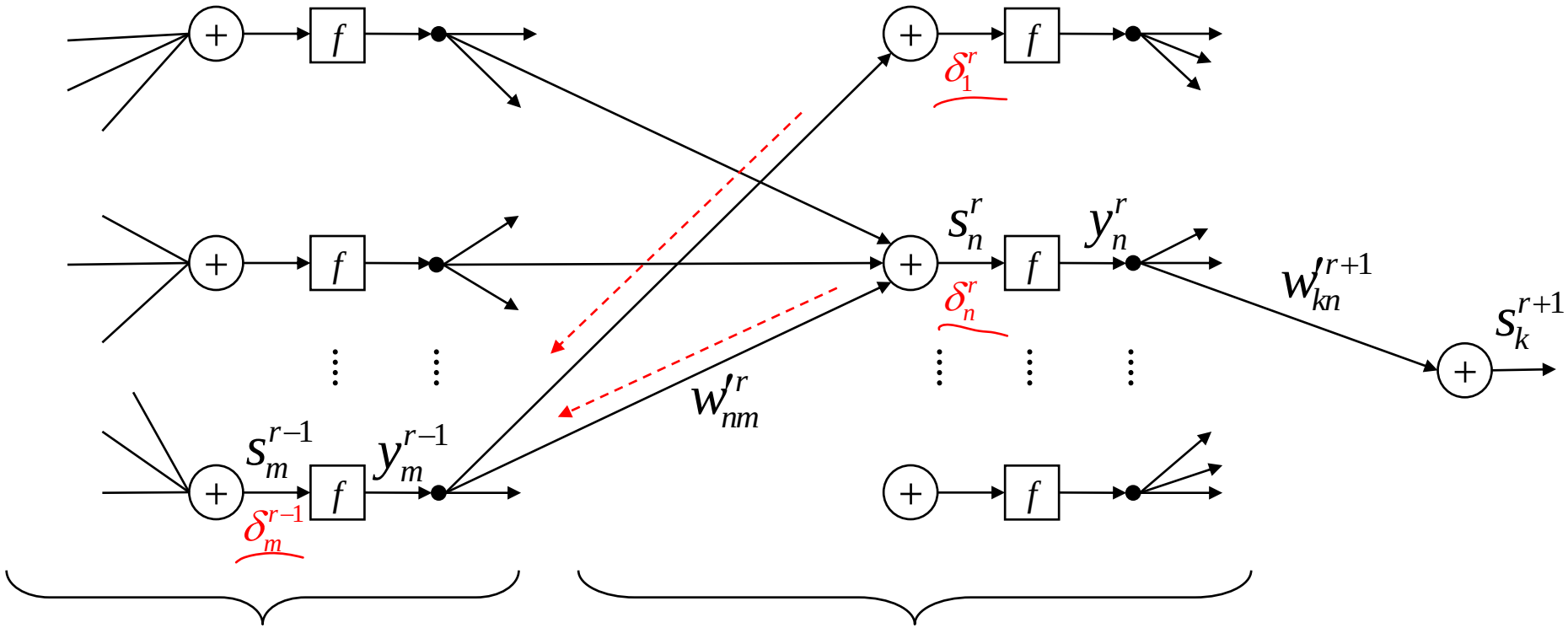
- Partielle Ableitungen für *eine* Schicht:
  Für die *r*-te Schicht gilt:

$$y_n^r = \psi(s_n^r) = \psi\left(\sum_{m=0}^{N^r} w'^r_{nm} x'^r_m\right) \quad \text{wobei:} \quad x'^r_m = y_m^{r-1}$$

x'^r_m m-ter Eingang von Schicht r

y_n^r n-ter Ausgang von Schicht r

Definition der Variablen zweier Schichten für den Backpropagation-Algorithmus



- Wir berechnen zunächst die Wirkung einer Variation der letzten verdeckten Schicht auf die Ausgangsschicht $r=H$.
  Unter Verwendung der partiellen Ableitungen der Gütefunktion J (eigentlich J_j, aber j wird der Einfachheit halber weggelassen) und unter Anwendung der *Kettenregel* erhält man:

$$\frac{\partial J}{\partial w'^{H}_{nm}} = \frac{\partial J}{\partial s^{H}_{n}} \cdot \frac{\partial s^{H}_{n}}{\partial w'^{H}_{nm}} = -\delta^{H}_{n} \underbrace{\frac{\partial s^{H}_{n}}{\partial w'^{H}_{nm}}}_{=y^{H-1}_{m}}$$

linearer Zusammenhang

- unter Einführung der *Empfindlichkeit* von Zelle n

$$\delta_n = -\frac{\partial J}{\partial s_n}$$

Abltg. der Aktiv.-Fkt.

- ergibt sich:

$$\delta^{H}_{n} = -\frac{\partial J}{\partial s^{H}_{n}} = -\frac{\partial J}{\partial \hat{y}^{H}_{n}} \cdot \frac{\partial \hat{y}^{H}_{n}}{\partial s^{H}_{n}} = \underbrace{(y_n - \hat{y}_n)}_{\frac{\partial\left(\frac{1}{2}\sum_{k=1}^{N^H}(\hat{y}_k - y_k)\right)^2}{\partial \hat{y}_n}} f'(s^{H}_{n})$$

- und damit schließlich für das Update der Gewichte:

$$w_{neu} = w_{alt} + \Delta w \quad \text{mit} \quad \Delta w = -\alpha \frac{\partial J}{\partial w}$$

$$\Delta w'^{H}_{nm} = \alpha \delta^{H}_{n} y^{H-1}_{m} = \alpha (y_n - \hat{y}_n) f'(s^{H}_{n}) y^{H-1}_{m}$$

- Für alle anderen verdeckten Schichten $r < H$ sind die Überlegungen etwas komplexer. Wegen der Abhängigkeit der Schichten untereinander, beeinflussen die Werte von s_m^{r-1} alle Elemente s_n^r der nachfolgenden Schicht. Unter Anwendung der *Kettenregel* erhält man:

$$\frac{\partial J}{\partial w'^{r}_{nm}} = \underbrace{\frac{\partial J}{\partial s^r_n}}_{-\delta^r_n} \underbrace{\frac{\partial s^r_n}{\partial w'^{r}_{nm}}}_{y^{r-1}_m} \quad \text{und weiter:} \quad \underbrace{\frac{\partial J}{\partial s^r_n}}_{-\delta^r_n} = \sum_k \underbrace{\frac{\partial J}{\partial s^{r+1}_k}}_{-\delta^{r+1}_k} \frac{\partial s^{r+1}_k}{\partial s^r_n}$$

- Mit

$$\frac{\partial s^{r+1}_k}{\partial s^r_n} = \frac{\partial \left(\sum_n w'^{r+1}_{kn} \underbrace{f(s^r_n)}_{y^r_n} \right)}{\partial s^r_n} = w'^{r+1}_{kn} f'(s^r_n)$$

- ergibt sich:

$$\boxed{\delta^r_n = f'(s^r_n) \left[\sum_k w'^{r+1}_{kn} \delta^{r+1}_k \right]}$$

- D.h. die Fehler „backpropagieren“ von der Ausgangs- zu niederen Schichten!

- Man durchläuft alle Lernstichproben, ohne irgendwelche Veränderungen an den Gewichtskoeffizienten, und man erhält den Gradienten $\nabla\mathbf{J}$ durch Mittelung über die $\nabla\mathbf{J}_j$. Beide Größen können zum Aufdaten der Parametermatrix $\mathbf{W}'$ des Perceptrons verwendet werden:

$$\mathbf{W}'_{k+1} = \mathbf{W}'_k - \alpha\nabla\mathbf{J}_j \qquad \text{individuelles Lernen}$$

$$\mathbf{W}'_{k+1} = \mathbf{W}'_k - \alpha\nabla\mathbf{J} \qquad \text{kumulatives Lernen}$$

- Die Gradienten $\nabla\mathbf{J}_j$ und $\nabla\mathbf{J}$ unterscheiden sich. Der zweite ist der Mittelwert des ersten ($\nabla\mathbf{J} = E\{\nabla\ \mathbf{J}_j\}$), oder: der erste stellt einen zufälligen Wert des zweiten dar.

Die gesamte individuelle Lernprozedur besteht aus den folgenden Verarbeitungsschritten:

- Wähle vorläufige Werte für die Koeffizientenmatrizen $\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^H$ aller Schichten
- Eine neue Beobachtung $[\mathbf{x}'_j, \mathbf{y}_j]$ sei gegeben
- Berechne die Diskriminierungsfunktion $\hat{\mathbf{y}}_j$ aus der gegebenen Stichprobe $\mathbf{x}'_j$ und den momentanen Gewichtsmatrizen (Vorwärtsrechnung)
- Berechne den Fehler zwischen Schätzung $\hat{\mathbf{y}}$ und Zielvektor $\mathbf{y}$:

$$\Delta\mathbf{y} = \hat{\mathbf{y}} - \mathbf{y}$$

- Berechne den Gradienten $\nabla\mathbf{J}$ bzgl. aller Perceptron-Gewichte (error backpropagation). Berechne dazu zuerst δ_n^H der Ausgangsschicht gemäß:

$$\delta_n^H = -\frac{\partial J}{\partial s_n^H} = -\frac{\partial J}{\partial \hat{y}_n^H} \cdot \frac{\partial \hat{y}_n^H}{\partial s_n^H} = (y_n - \hat{y}_n) f'(s_n^H)$$

- und berechne daraus rückwärts alle Werte der niederen Schichten gemäß:

$$\delta_m^{r-1} = f'(s_m^{r-1}) \left[\sum_n w_{nm}^r \delta_n^r \right] \quad \text{für} \quad r = H, H-1, \ldots, 2$$

- unter Beachtung der ersten Ableitung der Sigmoidfunktion $f(s)=\psi(s)$:

$$f'(s)=\frac{d\psi(s)}{ds}=\psi(s)(1-\psi(s))=y(1-y)$$

bzw: $f'(s_m^r)=y_m^r(1-y_m^r)$

- Korrigiere (parallel) alle Perceptron-Gewichte gemäß:

$$w'^r_{nm} \leftarrow w'^r_{nm}-\alpha\frac{\partial J}{\partial w'^r_{nm}}=w'^r_{nm}+\alpha\delta_n^r y_m^{r-1} \quad \text{für} \quad \begin{array}{l} r=1,2,\ldots H \\ n=1,2,\ldots N^H \\ m=1,2,\ldots M^H=N^{H-1} \end{array}$$

- Beim individuellen Lernen werden die Gewichte $\{w'^{h}_{nm}\}$ mit $\nabla\mathbf{J}_j$ für jede Stichprobe korrigiert, wohingegen beim kumulativen Lernen die gesamte Lernstichprobe durchgearbeitet werden muss, um den gemittelten Gradienten $\nabla\mathbf{J}$ aus der Sequenz der $\{\nabla\mathbf{J}_j\}$ zu ermitteln, bevor die Gewichte korrigiert werden können gemäß:

$$w'^r_{nm} \leftarrow w'^r_{nm}+\sum_i \alpha\delta_n^r(i)y_m^{r-1}(i) \quad \text{für} \quad \begin{array}{l} r=1,2,\ldots H \\ n=1,2,\ldots N^H \\ m=1,2,\ldots M^H=N^{H-1} \end{array}$$

Backpropagation-Algorithmus in Matrixschreibweise

- Wähle vorläufige Werte für die Koeffizientenmatrizen $\mathbf{W}^1, \mathbf{W}^2, \ldots, \mathbf{W}^H$ aller Schichten
- Eine neue Beobachtung $[\mathbf{x}'_j, \mathbf{y}_j]$ sei gegeben
- Berechne die Diskriminierungsfunktion $\hat{\mathbf{y}}_j$ aus der gegebenen Stichprobe $\mathbf{x}'_j$ und den momentanen Gewichtsmatrizen (Vorwärtsrechnung). Speichere alle Werte von $\mathbf{y}^r$ und $\mathbf{s}^r$ in allen Zwischenschichten $r = 1,2,\ldots,H$.
- Berechne den Fehler zwischen Schätzung $\hat{\mathbf{y}}$ und Zielvektor $\mathbf{y}$ am Ausgang:

$$\Delta\mathbf{y} = \hat{\mathbf{y}} - \mathbf{y}$$

- Berechne den Gradienten $\nabla\mathbf{J}$ bzgl. aller Perceptron-Gewichte (error backpropagation). Berechne dazu zuerst $\boldsymbol{\delta}^H$ der Ausgangsschicht gemäß:

$$\boldsymbol{\delta}^H = -\frac{\partial J}{\partial \mathbf{s}^H} = -\frac{\partial J}{\partial \hat{\mathbf{y}}^H} \cdot \frac{\partial \hat{\mathbf{y}}^H}{\partial \mathbf{s}^H} = (\mathbf{y} - \hat{\mathbf{y}}) \circ f'(\mathbf{s}^H)$$

- und berechne daraus rückwärts alle Werte der niederen Schichten gemäß:

$$\boldsymbol{\delta}^{r-1} = f'(\mathbf{s}^{r-1}) \circ (\mathbf{W}^{r^T} \boldsymbol{\delta}^r) \quad \text{für} \quad r = H, H-1, \ldots, 2$$

- unter Beachtung der ersten Ableitung der Sigmoidfunktion $f(s)=\psi(s)$:

$$f'(s)=\frac{d\psi(s)}{ds}=\psi(s)(1-\psi(s))=y(1-y)$$

bzw: $f'(s_m^r)=y_m^r(1-y_m^r)$

- ***Individuelles Lernen***: Korrigiere (parallel) alle Perceptron-Gewichtsmatrizen mit $\nabla\mathbf{J}_j$ für jede Stichprobe gemäß:

$$\mathbf{W}'^r \leftarrow \mathbf{W}'^r - \alpha\frac{\partial J}{\partial \mathbf{W}'^r} = \mathbf{W}'^r + \alpha\boldsymbol{\delta}^r\mathbf{y}^{(r-1)^T} \quad \text{für } r=1,2,\ldots H$$

- ***Kumulatives Lernen***: es muss die gesamte Lernstichprobe durchgearbeitet werden , um den gemittelten Gradienten $\nabla\mathbf{J}$ aus der Sequenz der $\{\nabla\mathbf{J}_j\}$ zu ermitteln, bevor die Gewichte korrigiert werden können gemäß:

$$\mathbf{W}'^r \leftarrow \mathbf{W}'^r + \alpha\sum_j \boldsymbol{\delta}_j^r\mathbf{y}_j^{(r-1)^T} \quad \text{für } r=1,2,\ldots H$$

Eigenschaften:

- Der Backpropagation-Algorithmus ist einfach zu implementieren, aber sehr rechenaufwendig, insbesondere wenn die Koeffizientenmatrix groß ist, was zur Folge hat, dass auch die Lernstichprobe entsprechend groß sein muss. Nachteilig ist weiterhin die Abhängigkeit des Verfahrens von den Startwerten der Gewichte, dem Korrekturfaktor α und die Reihenfolge, in der die Stichproben abgearbeitet werden.
- Wie beim rekursiven Trainieren des Polynomklassifikators bleiben lineare Abhängigkeiten in den Merkmalen unberücksichtigt.
- Positiv zu vermerken ist, das der Gradientenalgorithmus auch für sehr große Probleme verwendet werden kann.
- Die Dimension der Koeffizientenmatrix **W** ergibt sich aus den Dimensionen des Eingangsvektors, der verdeckten Schichten, sowie des Ausgangsvektors (N,N^1,..., N^{H-1},K) zu:

$$T = \dim(\mathbf{W}) = \sum_{h=1}^{H} (N^{h-1}+1)N^h \quad \text{mit } N^0 = N \text{ und } N^H = K$$

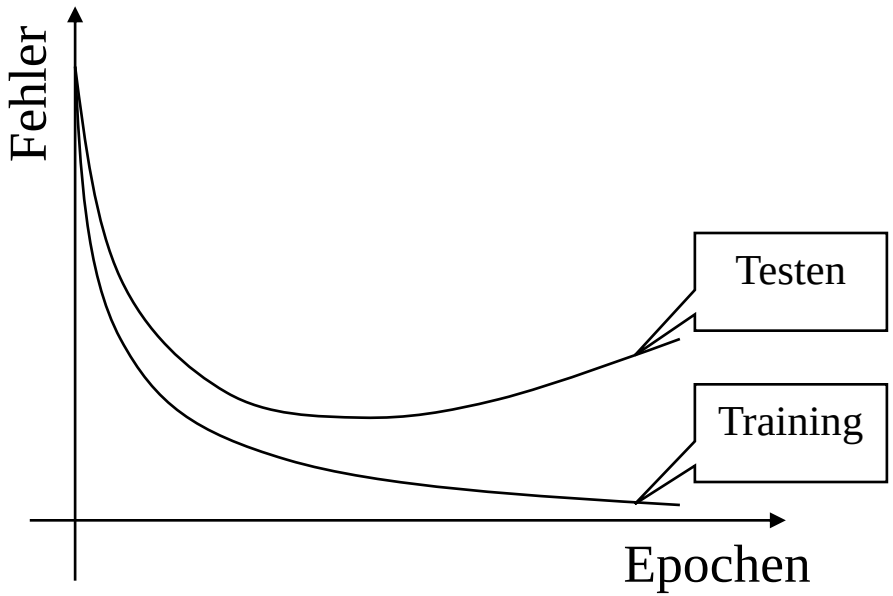
$N = \dim(\mathbf{x})$ Merkmalsraum

$K = \dim(\hat{\mathbf{y}})$ Anzahl der Klassen

Zur Dimensionierung des Netzes

- Die Überlegungen bei dem Entwurf eines mehrschichtigen Perceptrons mit Schwellwertfunktionen (σ bzw. sign) geben eine gute Vorstellung, wieviele Hidden-Layer und wieviele Neuronen man für ein MLPC verwenden sollte für das Backpropagation-Lernen (vorausgestzt, man hat eine gewisse Vorstellung über die Verteilung der Cluster).
- Das Netz sollte so einfach wie nur möglich gewählt werden. Mit höherer Dimension steigt die Gefahr des overfitting, gepaart mit einem Verlust an Generalisierungsfähigkeit und zugleich werden viele Nebenmaxima zugelassen, wo der Algorithmus hängen bleiben kann!

- Bei einem vorgegebenen Stichprobenumfang, sollte die Lernphase bei einem bestimmten Punkt abgebrochen werden. Danach wird das Netz zu sehr an die vorhandenen Daten angepasst (overfitting) und verliert an Generalisierungsfähigkeit.



Zur Berechnungskomplexität des Backpropagation-Algorithmus

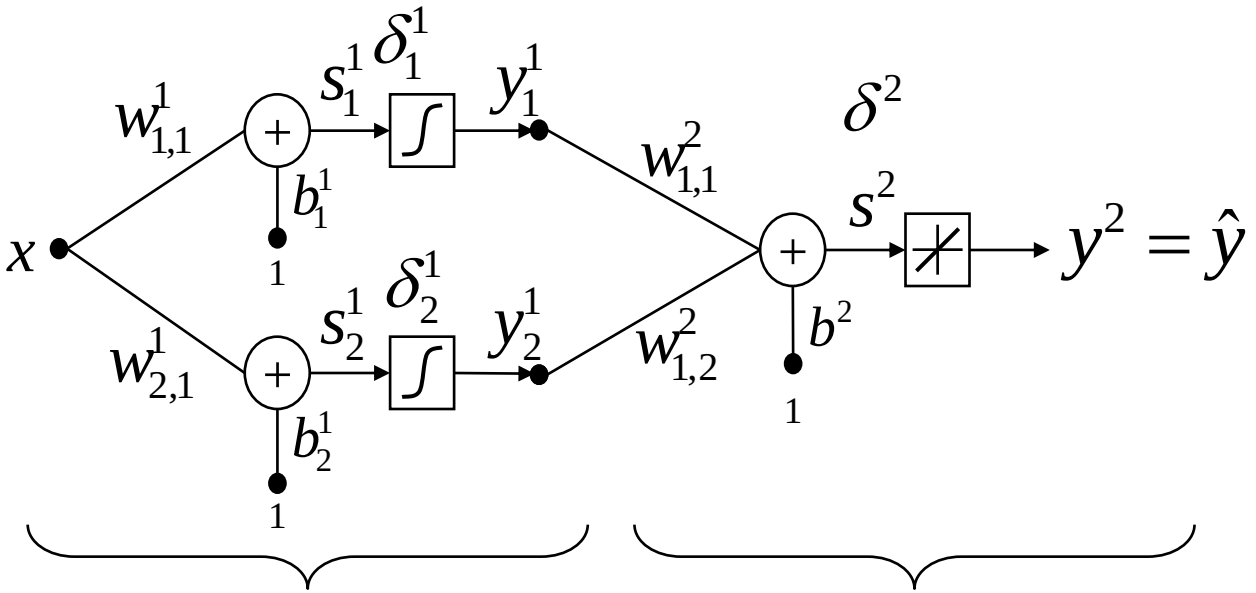
- Wenn T=dim(**W**) die Anzahl der Gewichte und Biasterme ist, so läßt sich einfach nachvollziehen, daß *O(T)* Berechnungsschritte für die Vorwärtssimulation benötigt werden, *O(T)* für das Backpropagieren des Fehlers und ebenso *O(T)* Operationen für die Korrektur der Gewichte, also erhält man insgesamt einen lineare Komplexität in der Anzahl der Gewichte: *O(T)*.
- Würde man den Gradienten durch finite Differenzen experimentell bestimmen (dazu ändert man jedes Gewicht inkrementell und ermittelt die Wirkung auf das Gütemaß durch Vorwärtsrechnung) durch Berechnung eines Differenzenquotienten gemäß (d.h. man macht sich nicht die Mühe der analytischen Auswertung):

$$\frac{\partial J}{\partial w_{ji}^r} = \frac{J(w_{ji}^r + \varepsilon) - J(w_{ji}^r)}{\varepsilon} + O(\varepsilon)$$

- so ergäben sich T Vorwärtsrechnungen der Komplexität *O(T)* und damit insgesamt eine Gesamtkomplexität von: $O(T^2)$

Backpropagation zum Trainieren eines zweischichtigen Netzwerks zur Funktionsapproximation

(Matlab-Demo: „Backpropagation Calculation)



$$\mathbf{y}^1 = \boldsymbol{\psi}(\mathbf{W}^1 x + \mathbf{b}^1) = \boldsymbol{\psi}(\mathbf{W}'^1 \mathbf{x}'^1) = \boldsymbol{\psi}(\mathbf{s}^1) \qquad \hat{y} = y^2 = (\mathbf{W}^2 \mathbf{y}^1 + b^2)$$

mit: $\mathbf{W}^1 = \begin{bmatrix} w_{1,1}^1 \\ w_{2,1}^1 \end{bmatrix}$ und $\mathbf{b}^1 = \begin{bmatrix} b_1^1 \\ b_2^1 \end{bmatrix}$ mit: $\mathbf{W}^2 = \begin{bmatrix} w_{1,1}^2 & w_{1,2}^2 \end{bmatrix}$

- Gesucht wird eine Approximation der Funktion

$$y(x) = 1 + \sin(\tfrac{\pi}{4} x) \quad \text{für} \quad -2 \le x \le 2$$

Backpropagation zum Trainieren eines zweischichtigen Netzwerks zur Funktionsapproximation

- Backpropagation: Für die Ausgangsschicht oder zweite Schicht mit linearer Aktivierungsfunktion ergibt sich wegen $f'=1$:

$$\delta^2 = (y - \hat{y})$$

- Backpropagation: Für die erste Schicht mit Sigmoid-Funktion ergibt sich: :

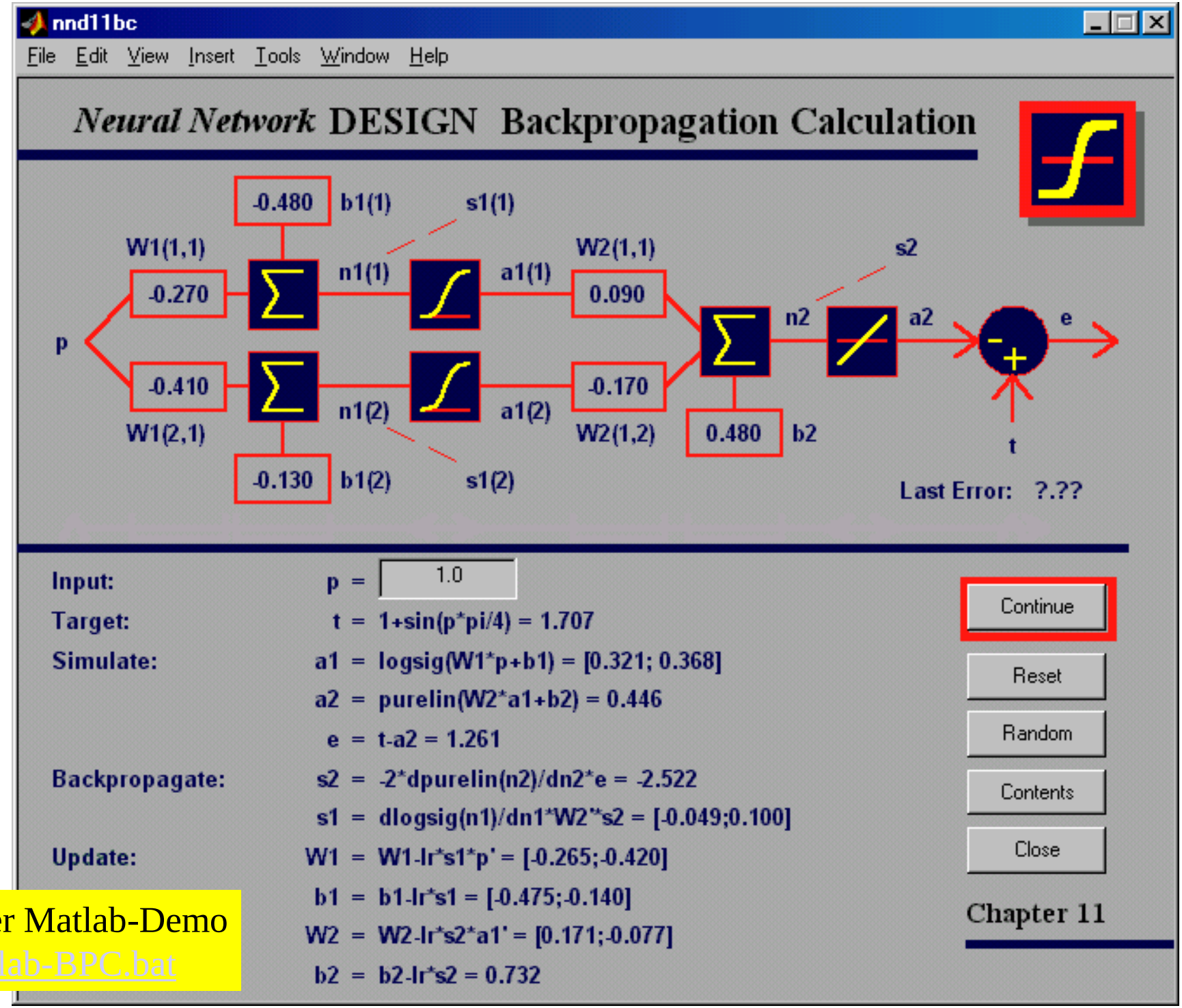
$$\delta_j^1 = f'(s_j^1)\left[w_{1,j}^2 \delta^2\right] = y_j^1(1-y_j^1)\left[w_{1,j}^2 \delta^2\right] \quad \text{für} \quad j = 1,2$$

- Korrektur der Gewichte in der Ausgangsschicht:

$$\Delta w_{1,j}^2 = \alpha \delta^2 y_j^1 \quad \text{und} \quad \Delta b^2 = \alpha \delta^2 \quad \text{für} \quad j = 1,2$$

- Korrektur der Gewichte in der Eingangsschicht:

$$\Delta w_{j,1}^1 = \alpha \delta_j^1 x \quad \text{und} \quad \Delta b_j^1 = \alpha \delta_j^1 \quad \text{für} \quad j = 1,2$$



Start der Matlab-Demo

matlab-BPC.bat

Backpropagation zum Trainieren eines zweischichtigen Netzwerks zur Funktionsapproximation

(Matlab-Demo, other NN, Backpropagation: „Function Approximation)

- Gesucht wird eine Approximation der Funktion

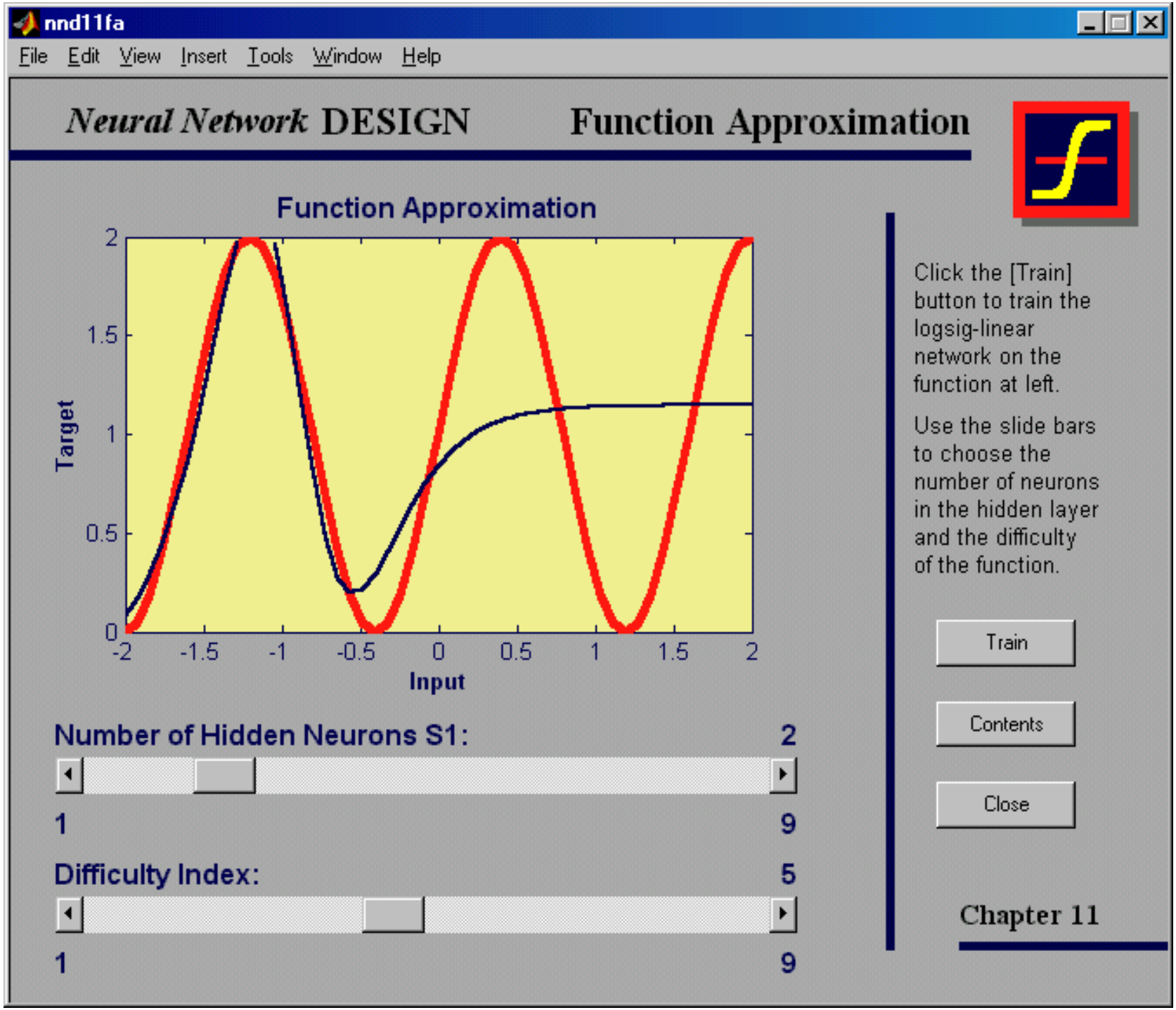
$$y(x) = 1 + \sin(i \cdot \tfrac{\pi}{4} x) \quad \text{für} \quad -2 \le x \le 2$$

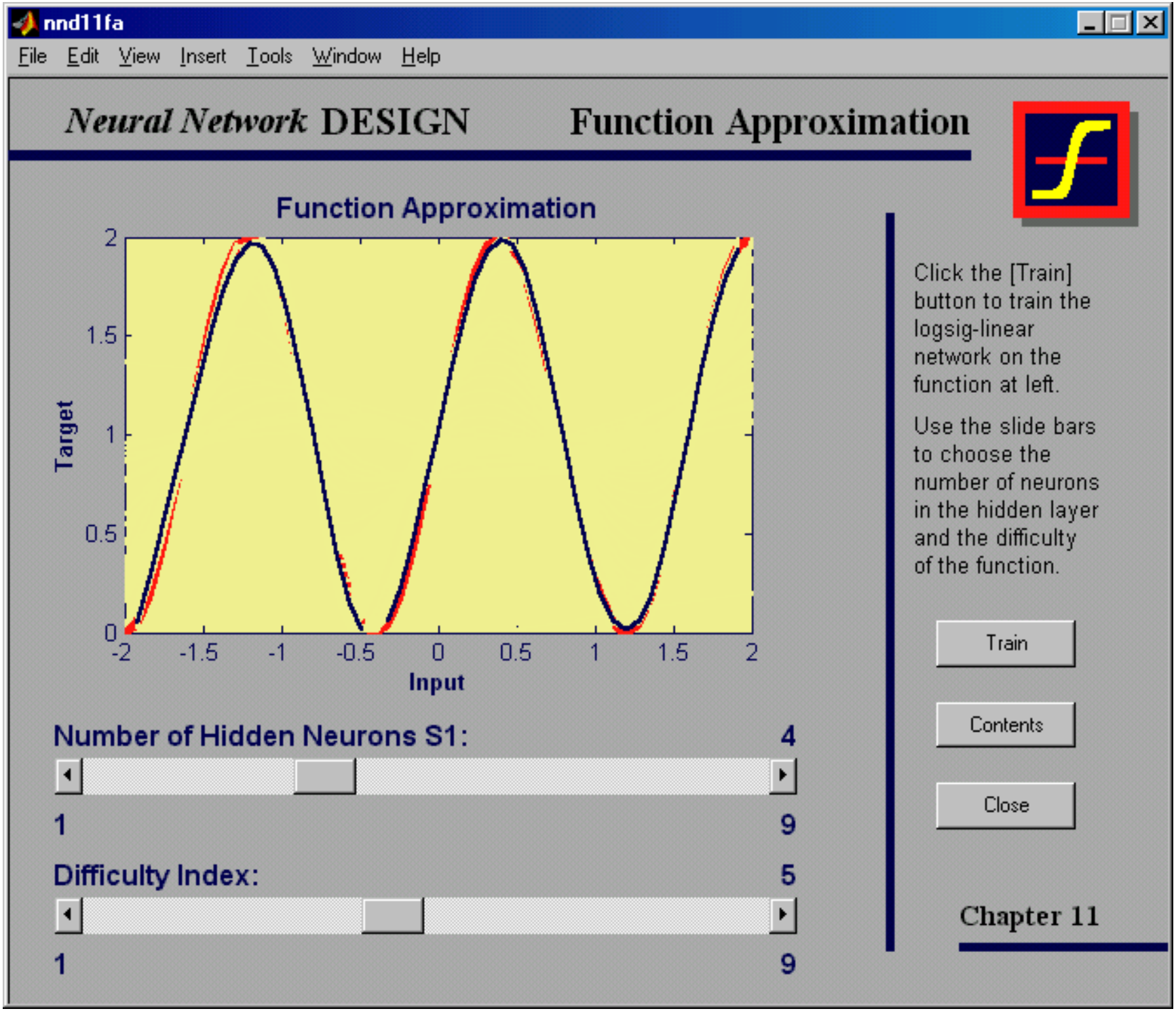
- Mit zunehmenden Wert von i (difficulty index) steigen die Anforderungen an das MLPC-Netzwerk. Die auf der Sigmoid-Funktion aufbauende Approximation benötigt mehr und mehr Schichten, um mehrere Perioden der Sinus-Funktion darzustellen.
- Problem: Konvergenz zu lokalen Minima, selbst in Fällen, wo das Netzwerk groß genug gewählt wird, um die Funktion zu approximieren
  - Das Netzwerk wird zu klein gewählt, so dass eine Approximation nur schlecht gelingt, d.h. es wird zwar das globale Minimum erreicht, aber dieses liefert noch keine gute Approximationsgüte (i=8 und Netzwerk 1-3-1)
  - Das Netzwerk wird groß genug gewählt, so dass eine gute Approximation möglich ist, aber es konvergiert nur gegen ein lokales Minimum (i=4 und Netzwerk 1-3-1)

$$\hat{y}(x) = \sum w_i \psi(w_j x - b_k)$$

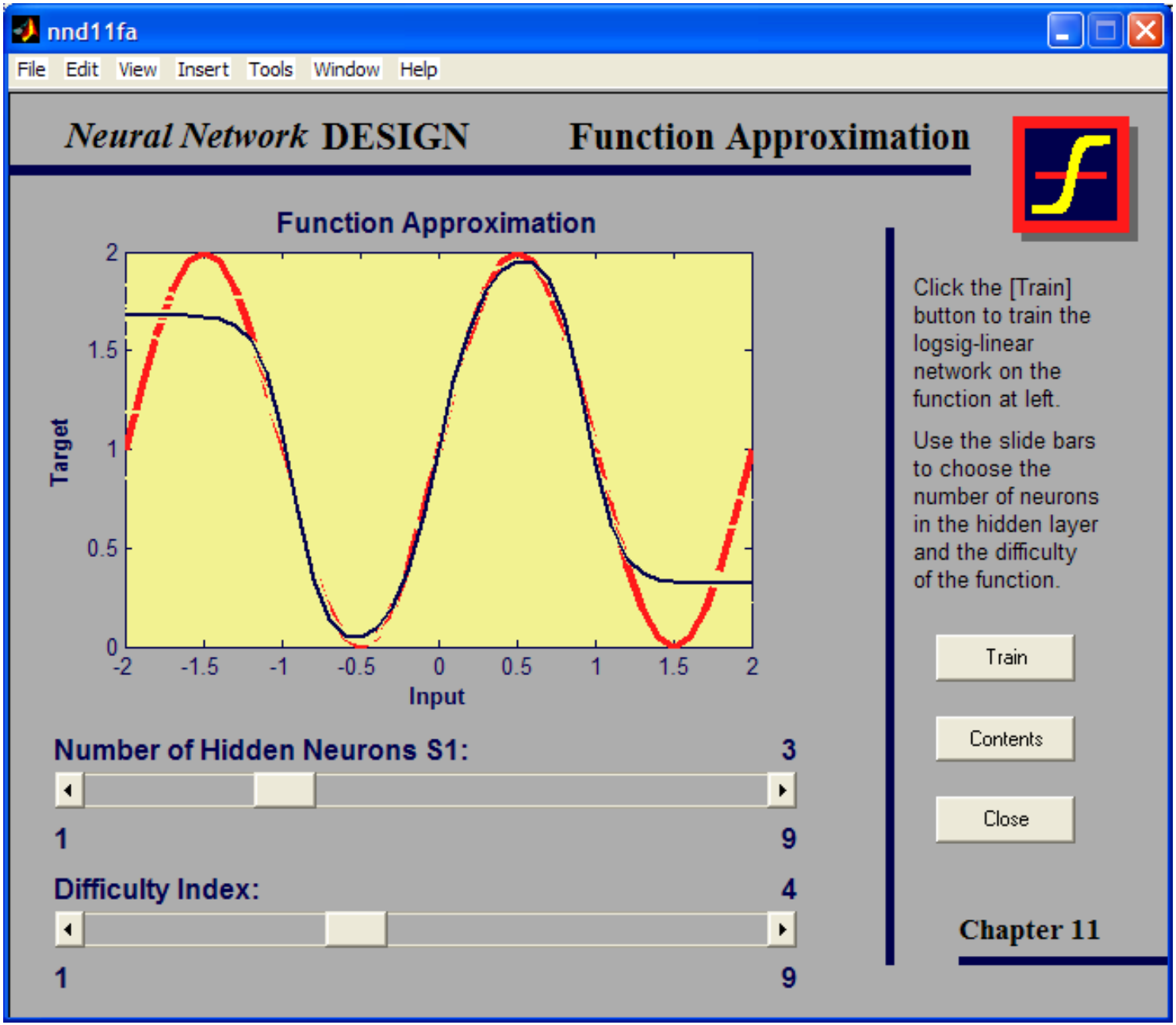
Reihe *linear* in den w_i, aber *nichtlinear* in den Parametern w_j,b_i

Start der Matlab-Demo
matlab-FA.bat


nnd11fa
File Edit View Insert Tools Window Help
Neural Network DESIGN
Function Approximation
Function Approximation
Target
Input
2
1.5
1
0.5
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
Number of Hidden Neurons S1:
2
1
9
Difficulty Index:
5
1
9
Click the [Train] button to train the logsig-linear network on the function at left.
Use the slide bars to choose the number of neurons in the hidden layer and the difficulty of the function.
Train
Contents
Close
Chapter 11


nnd11fa
File Edit View Insert Tools Window Help
Neural Network DESIGN
Function Approximation
Function Approximation
Target
Input
2
1.5
1
0.5
0
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
Number of Hidden Neurons S1:
4
1
9
Difficulty Index:
5
1
9
Click the [Train] button to train the logsig-linear network on the function at left.
Use the slide bars to choose the number of neurons in the hidden layer and the difficulty of the function.
Train
Contents
Close
Chapter 11

Modell groß genug, aber nur lokales Minimum



Generalisierungsfähigkeit des Netzwerkes

- Wir gehen davon aus, dass das Netzwerk für 11 Abtastpunkte trainiert wird

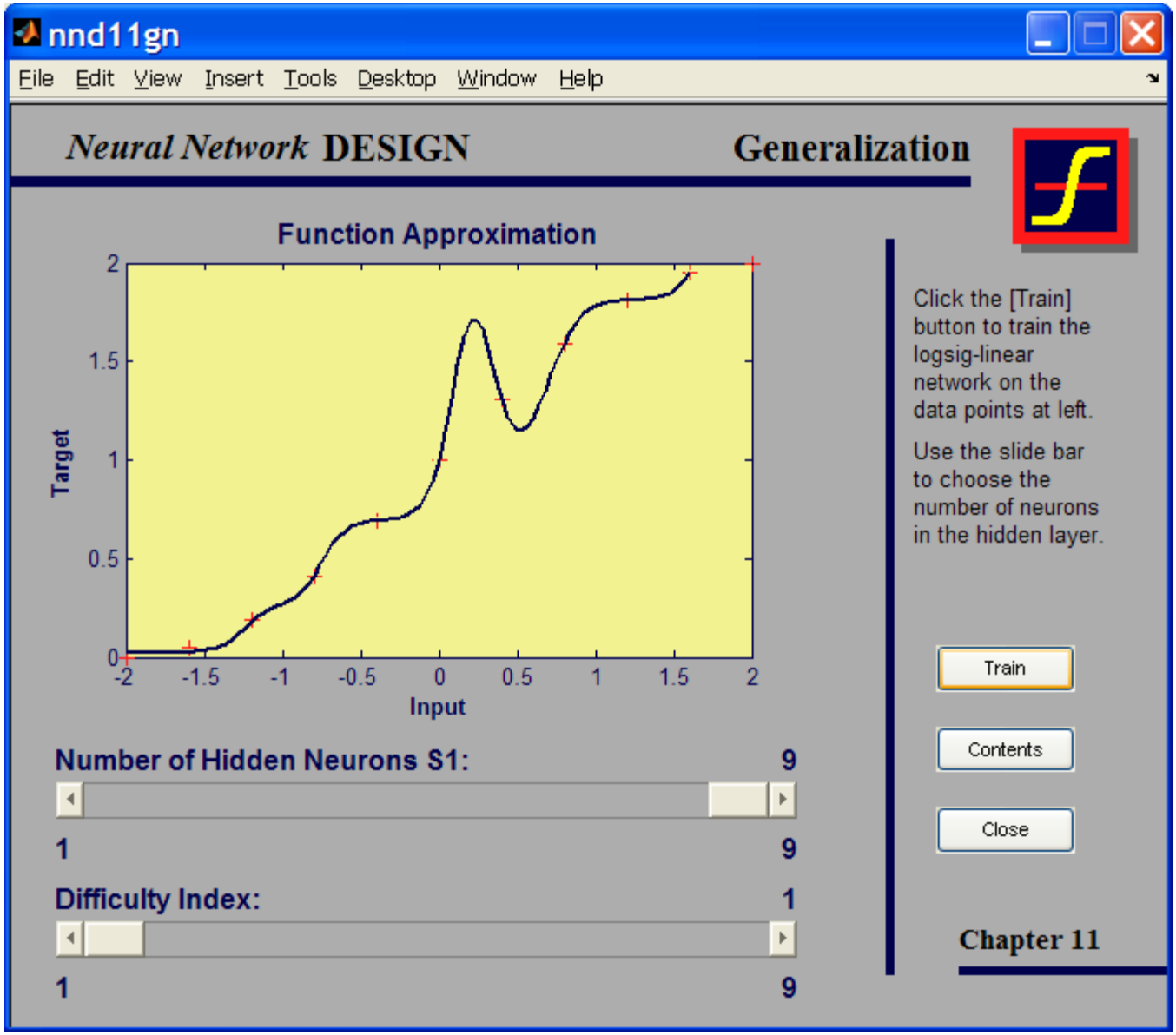
$$\{y_1, x_1\}, \{y_2, x_2\}, \ldots \{y_{11}, x_{11}\}$$

- Die Frage ist nun, wie gut das Netzwerk die Funktion für nicht gelernte Abtastpunkte approximiert in Abhängigkeit von der Komplexität des Netzwerkes

Start der Matlab-Demo (Hagan 11-21)
matlab-GENERAL.bat

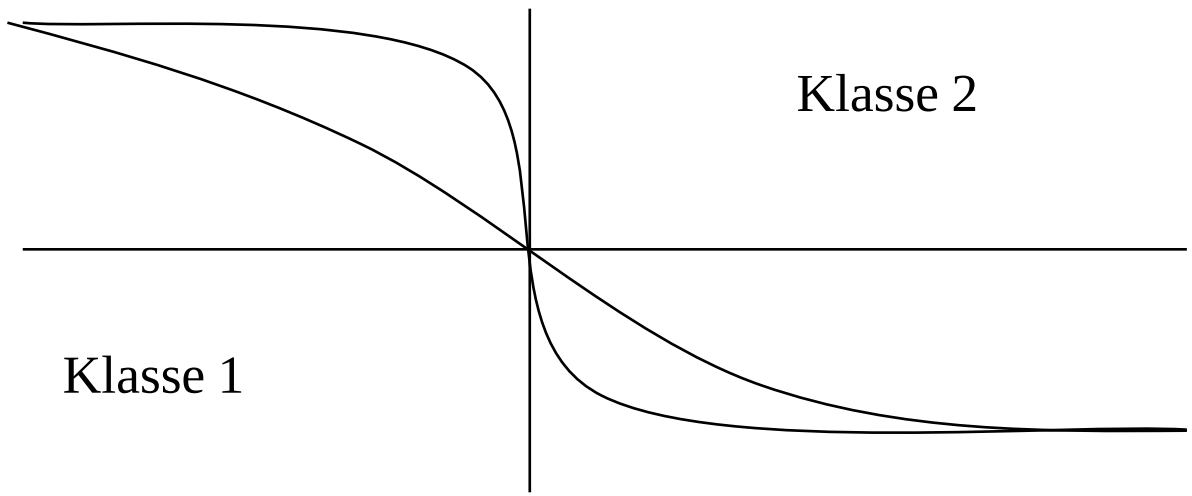
- Regel: Das Netzwerk sollte weniger Parameter haben als die zur Verfügung stehenden Ein-/Ausgangspaare

Typisches Overfitting

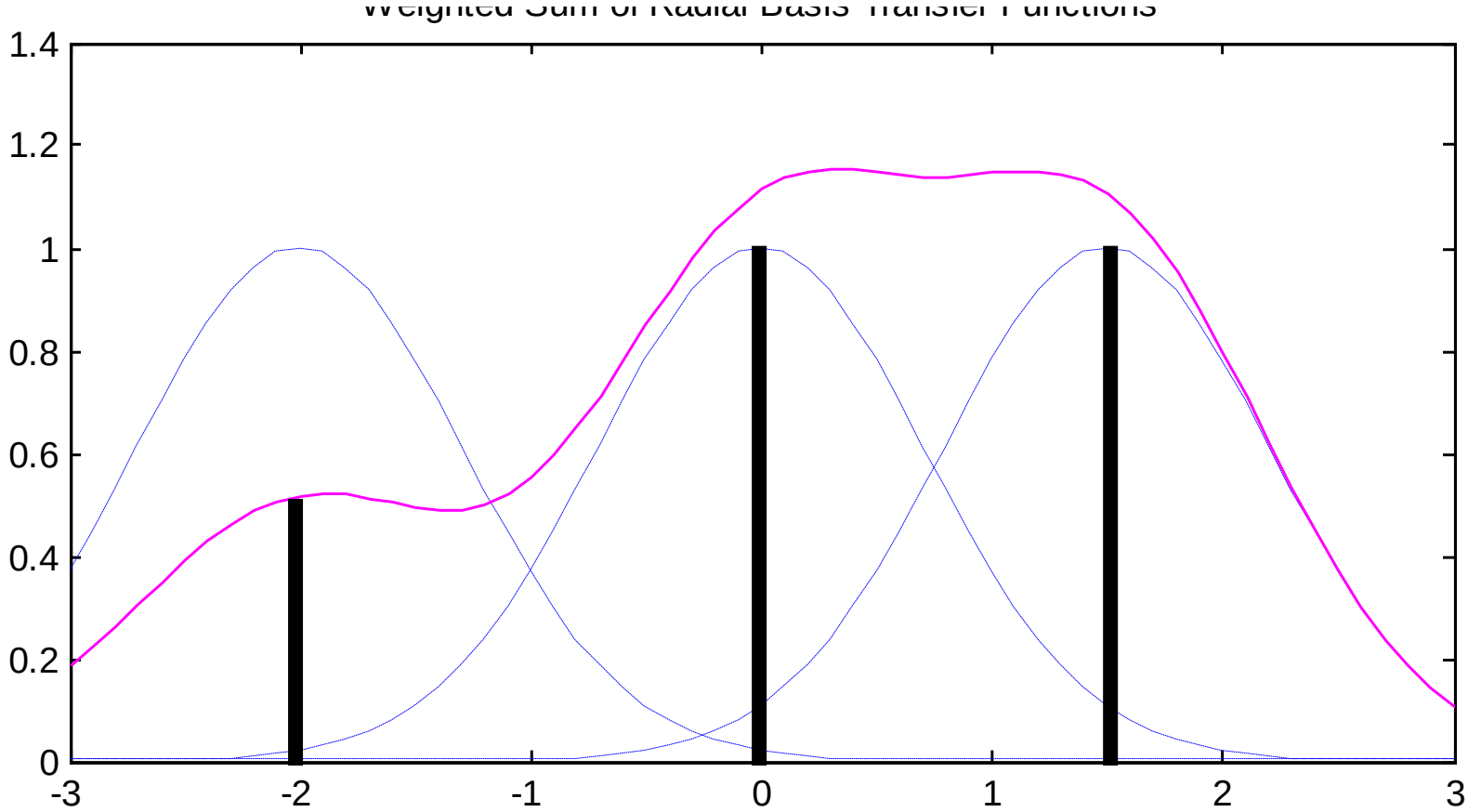


Verbesserung der Generalisierungsfähigkeit eines Netzes durch Hinzufügen additiver Störungen

- Stehen nur wenige Stichproben zur Verfügung, so kann die Generalisierungs-fähigkeit eines NN verbessert werden, indem man den Stichprobenumfang durch z.Bsp. normalverteilte Störungen verbreitert. D.h. man fügt weitere Stichproben hinzu, welche mit hoher Wahrscheinlichkeit in der unmittelbaren Nachbarschaft anzutreffen sind.
- Dadurch werden die Intraklassenbereiche verbreitert und die Grenzen zwischen den Klassen schärfer ausgebildet.

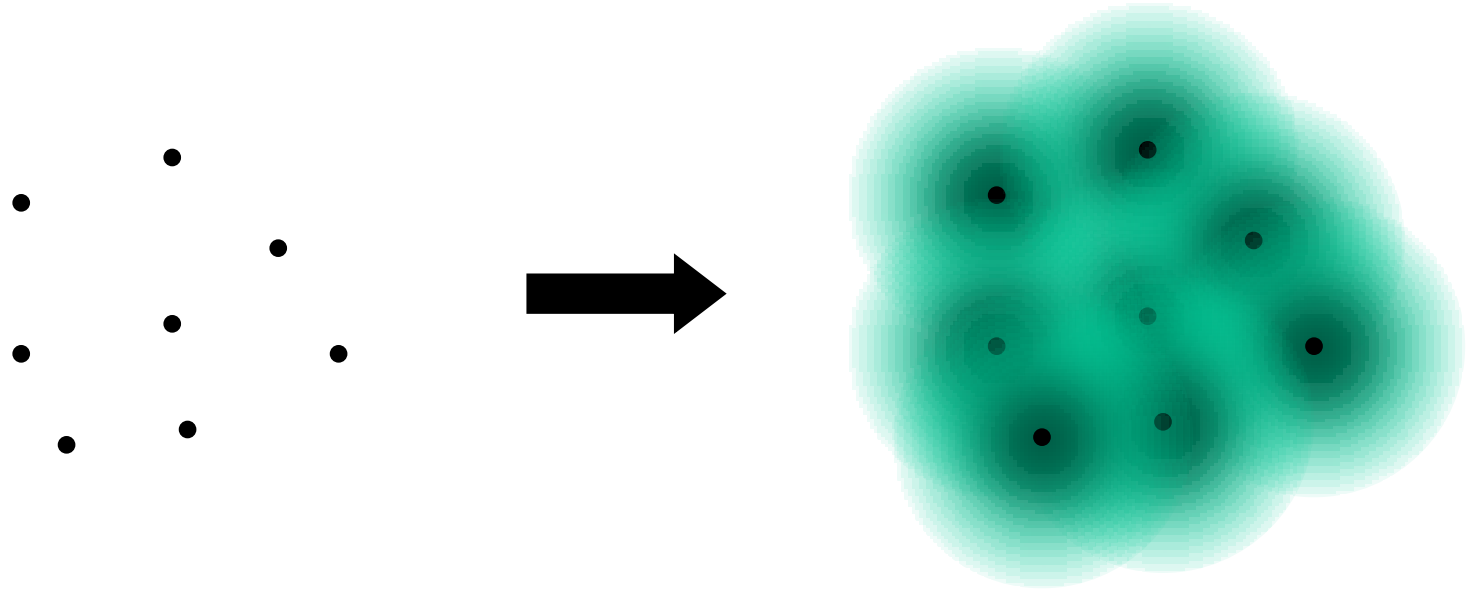


Interpolation durch Überlagerung von Normalverteilungen

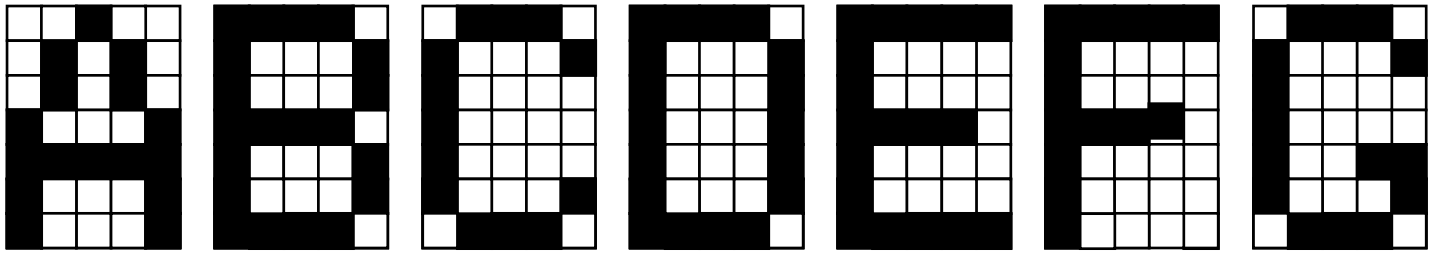


$$y(x) = \sum \alpha_i f_i(x - t_i)$$

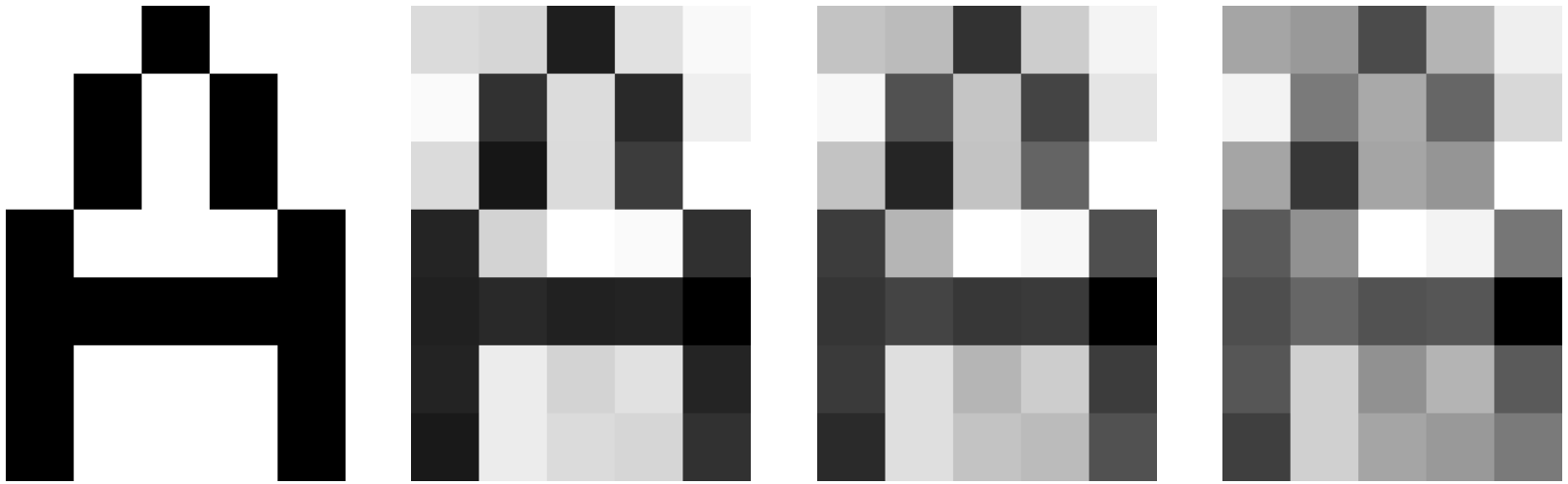
Interpolation durch Überlagerung von Normalverteilungen



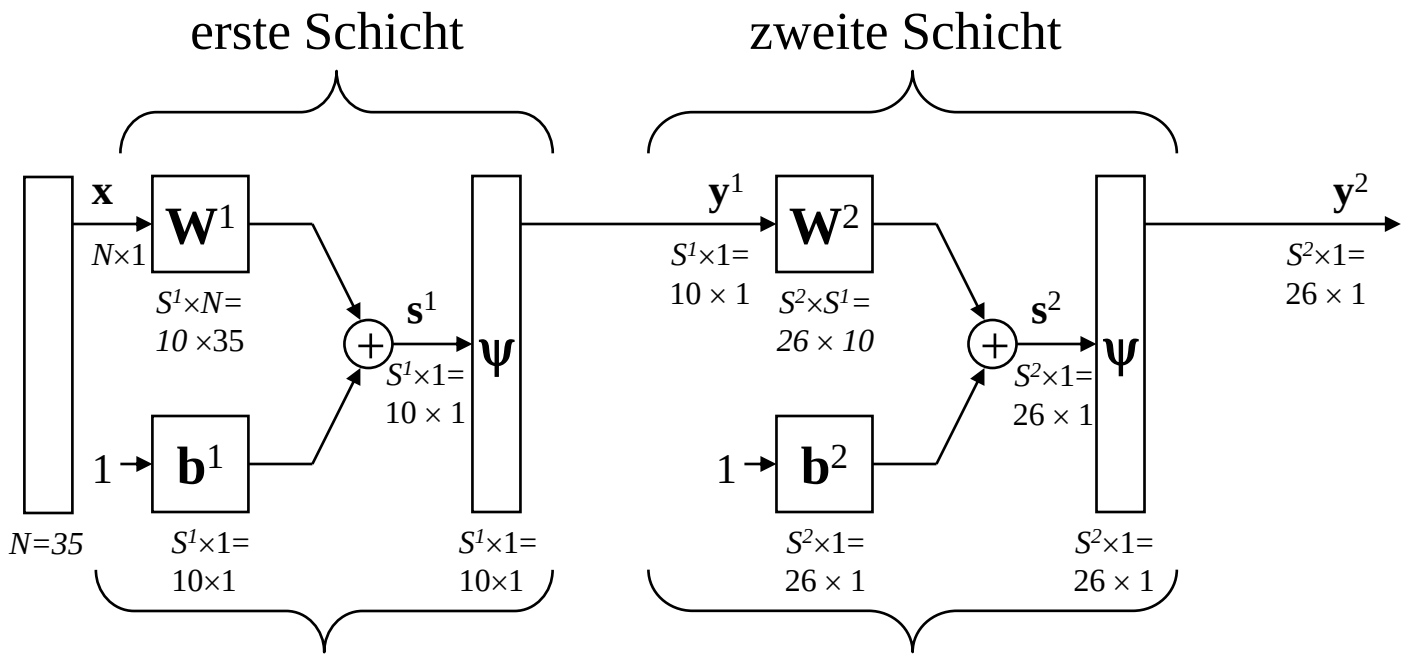
Zeichenerkennungsaufgabe für 26 Buchstaben der Größe 7×5



- mit graduell ansteigenden additiven Störungen:



Zweischichtiges Neuronales Netz für die Zeichenerkennung mit 35 Eingangswerten (Pixel), 10 Neuronen in der verdeckten Schicht und 26 Neuronen in der Ausgangsschicht



$\mathbf{y}^1=\mathbf{f}^1(\mathbf{W}^1\mathbf{x}+\mathbf{b}^1)$ $\mathbf{y}^2=\mathbf{f}^2(\mathbf{W}^2\mathbf{y}^1+\mathbf{b}^2)$

$$\mathbf{y}^2=\psi(W^2\,\psi(W^1x+\mathbf{b}^1)+\mathbf{b}^2)$$

Demo mit MATLAB

- Öffnen von Matlab
  - Demo in Toolbox Neural Networks
  - „Character recognition“ (command line)
  - Es werden zwei Netzwerke trainiert
    - Netzwerk 1 ohne Störungen
    - Netzwerk 2 mit Störungen
  - Auf einem unabhängigen Testdatensatz liefert Netzwerk 2 bessere Ergebnisse als Netzwerk 1

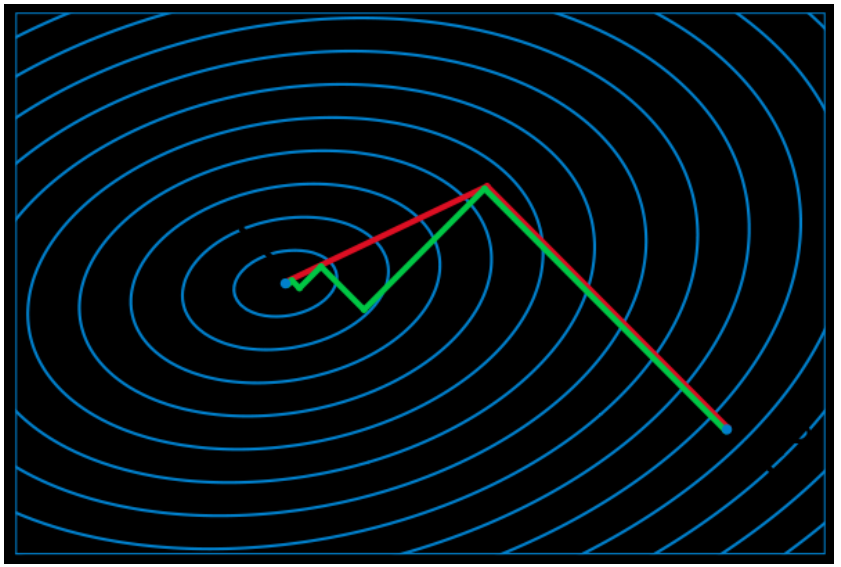
Start der Matlab-Demo
matlab-CR.bat

Möglichkeiten zur Beschleunigung der Adaption

Es handelt sich bei der Adaption von NN um ein allgemeines, *globales* Parameter-optimierungsproblem. Demgemäß können im Prinzip eine Vielzahl weiterer Optimierungsmethoden aus der numerischen Mathematik eingesetzt werden. Dies kann zu erheblichen Verbesserungen führen (Konvergenzgeschwindigkeit, Konvergenzbereich, Stabilität, Berechnungsaufwand).

I.a. lassen sich jedoch nur sehr schwer allgemeingültige Aussagen machen, da die Ergebnisse von Fall zu Fall sehr verschieden sein können und in der Regel Kompromisse in den Eigenschaften zu akzeptieren sind!

- Heuristische Verbesserungen des Gradientenalgorithmus (Schrittweitenkontrolle)
  - Gradientenalgorithmus (Steepest descent) mit Momentum-Term (update der Gewichte nicht nur abhängig vom Gradient, sondern auch vom vorherigen update)
  - Verwendung eines adaptiven Lernfaktors
- Konjugierter Gradientenalgorithmus



- Newton und Quasi-Newton-Algorithmen (Verwendung der zweiten Ableitungen der Fehlerfunktion z.B. in Form der Hesse-Matrix oder deren Schätzung)
  - Quickprop
  - Levenberg-Marquardt-Algorithmus

Taylorentwicklung der Gütefunktion in $\mathbf{w}$:

$$J(\mathbf{w}+\Delta\mathbf{w}) = J(\mathbf{w}) + \nabla\mathbf{J}^T\Delta\mathbf{w} + \tfrac{1}{2}\Delta\mathbf{w}^T\mathbf{H}\Delta\mathbf{w} + \text{Terme höherer Ordnung}$$

mit: $\nabla\mathbf{J} = \dfrac{\partial J}{\partial \mathbf{w}}$ Gradientenvektor

und: $\mathbf{H} = \left\{ \dfrac{\partial^2 J}{\partial w_i \partial w_j} \right\}$ Hesse-Matrix

- Pruning-Techniken. Man startet mit einem hinreichend großen Netzwerk und nimmt dann wieder Neuronen aus dem Netz, welche keinen oder nur geringen Einfluß auf die Gütefunktion haben und reduziert damit das Overfitting der Daten.

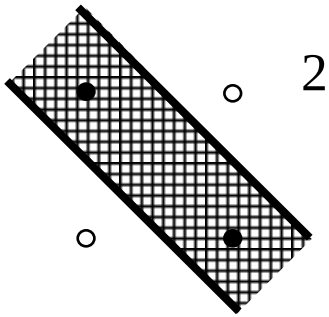
Demo mit MATLAB

(Demo von Theodoridis)

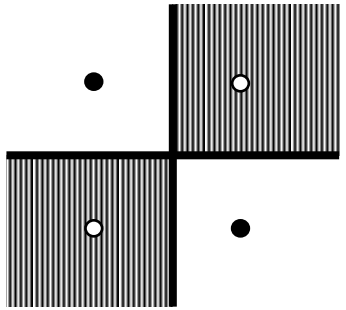
- C:\...\matlab\PR\startdemo$_0$
- Example 2 (XOR) mit (2-1 und 5-1)
- Example 3 mit dreischichtigem Netzwerk [5,5] (3000 Epochen, learning rate 0.7, momentum 0.5)

Start der Matlab-Demo
matlab-theodoridis.bat

Zwei Lösungen des XOR-Problems:



2-1 konvergiert



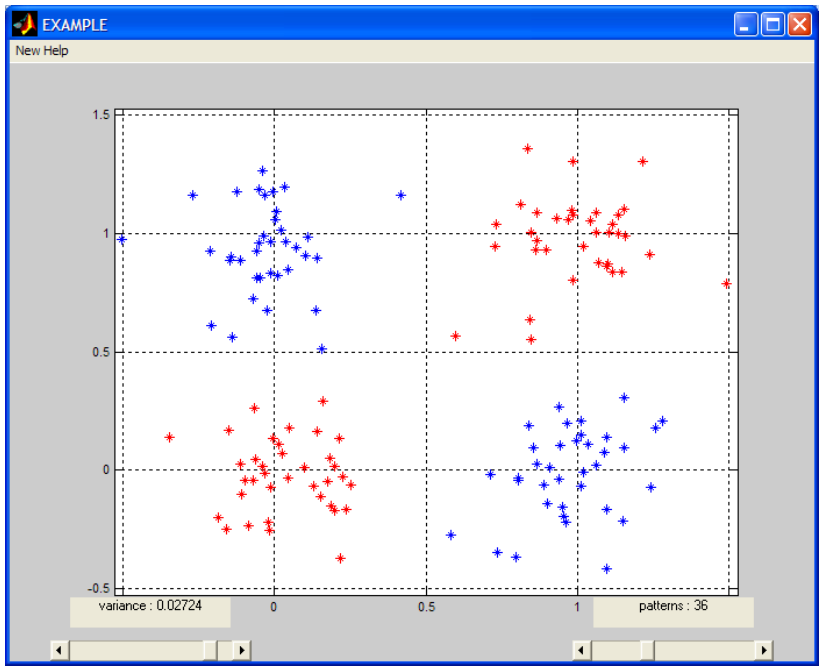
2-2-1 konvergiert leider nicht!

Konvexes Gebiet realisiert mit einem zweischichtigen Netz (2-1). Mögliche Fehlklassifikation bei Varianz:

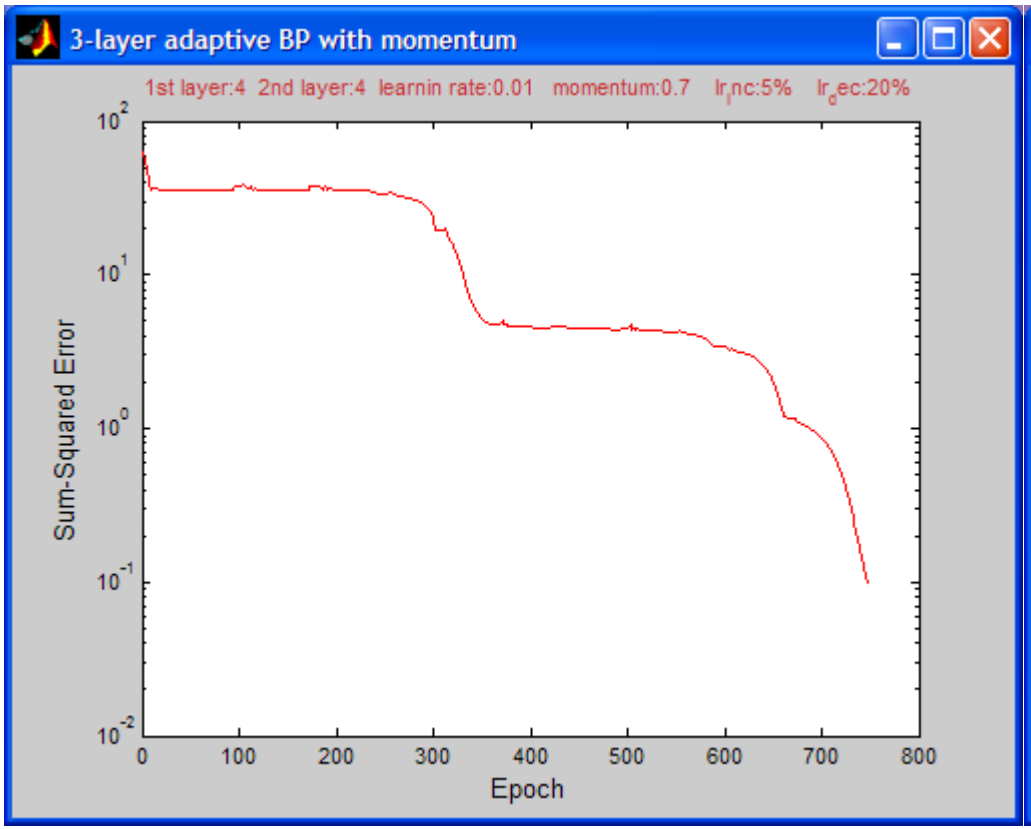
$$\|\mathbf{n}\| > \tfrac{1}{4}\sqrt{2} \approx 0{,}35$$

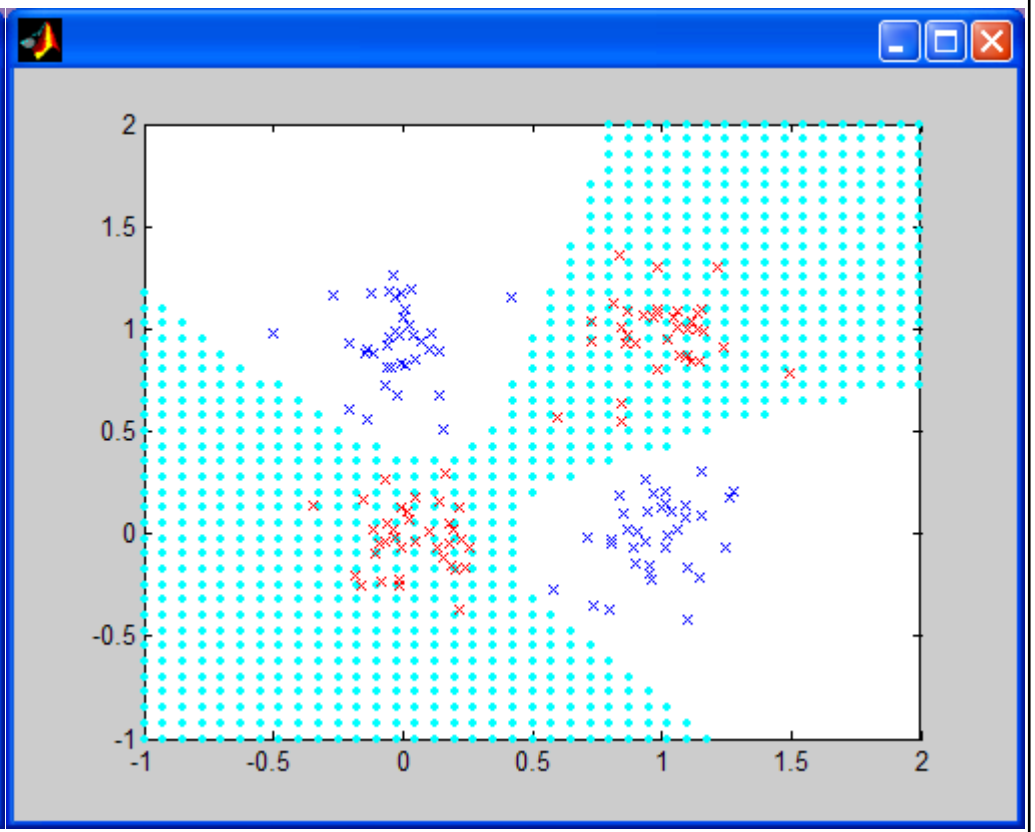
Vereinigung von 2 konvexen Gebieten realisiert mit einem dreischichtigen Netz (5-5-1). Mögliche Fehlklassifikation bei Varianz:

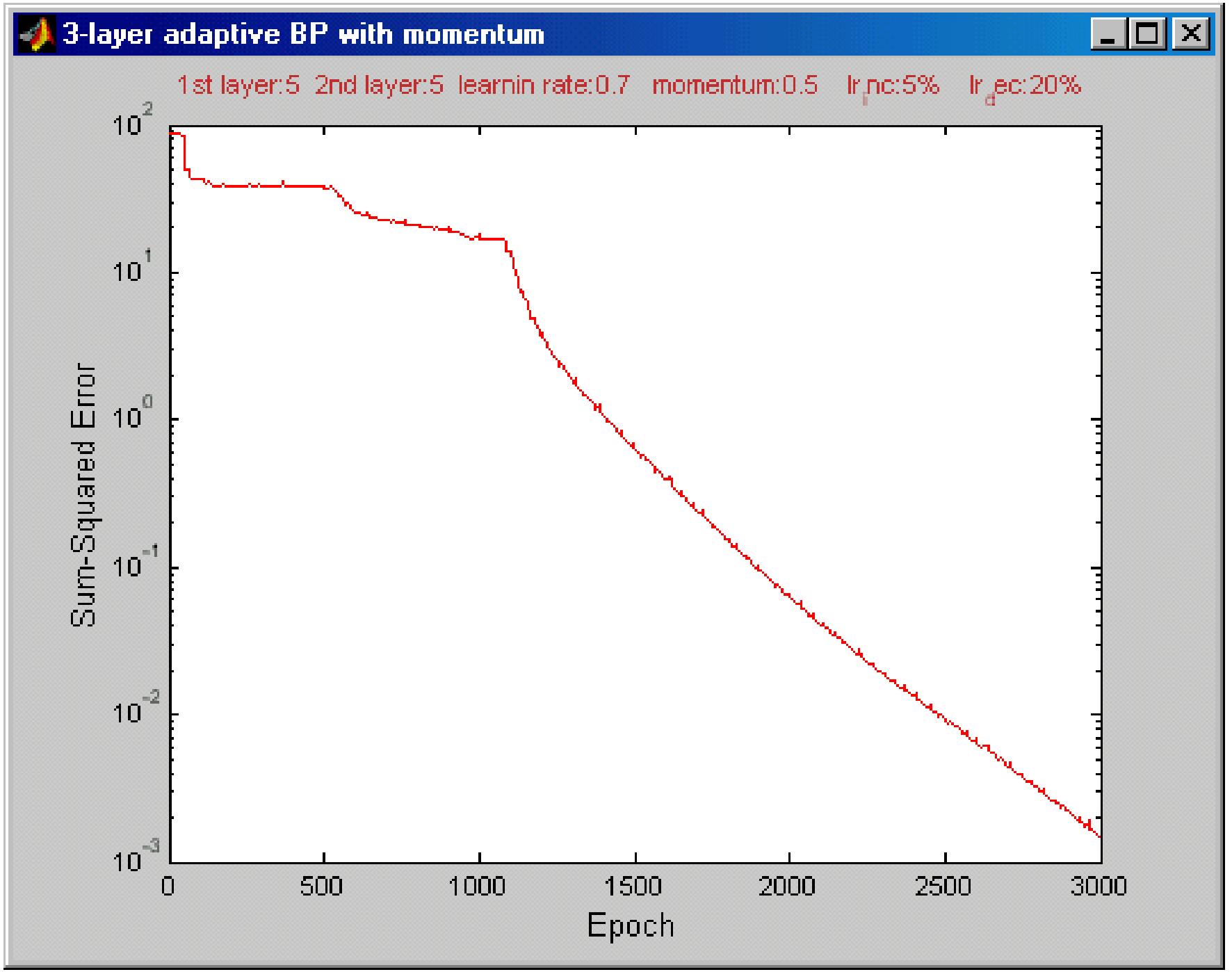
$$\|\mathbf{n}\| > 0{,}5$$

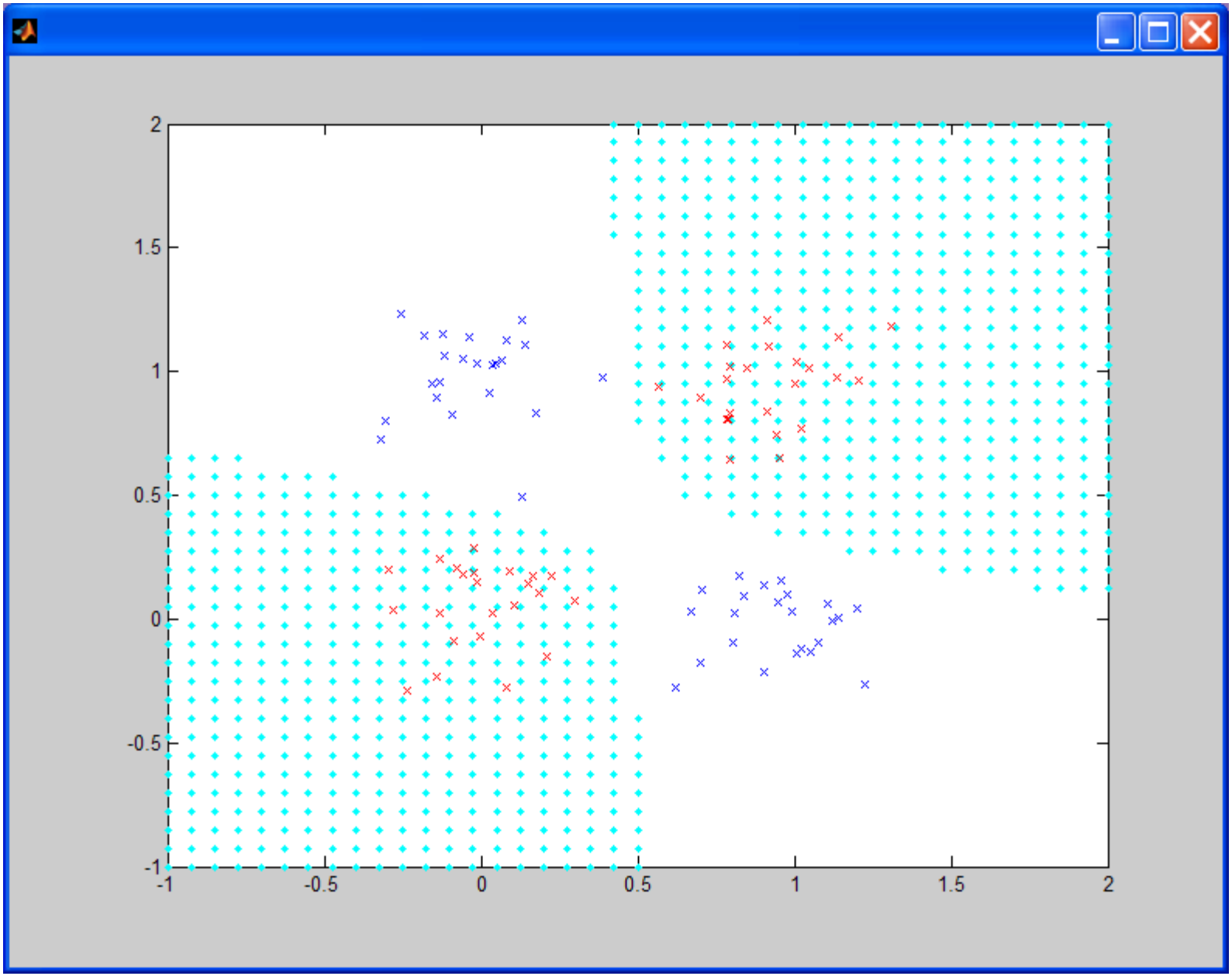


XOR mit hohem Rauschanteil um zweite Lösung zu provozieren! Mit zwei Geraden schafft mein keine fehlerfreie Konstellation. Der Gradient kann trotzdem verschwinden bei einem von Null verschiedenen Fehlermass, d.h. man ist in einem Nebenmaximum. Erst die richtige Lösung führt auf einen Fehler Null.

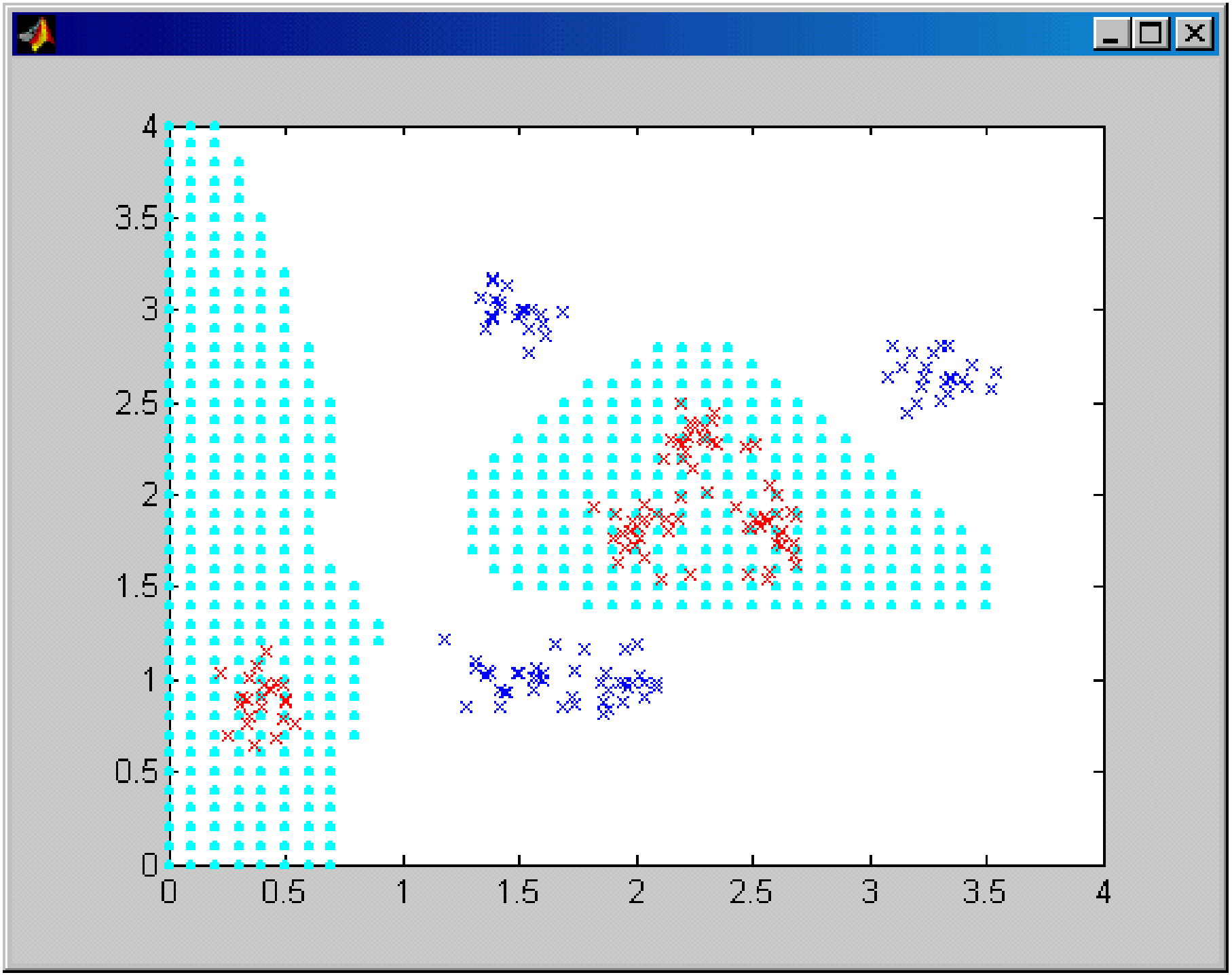





3-layer adaptive BP with momentum
1st layer:5 2nd layer:5 learnin rate:0.7 momentum:0.5 lr_inc:5% lr_dec:20%
Sum-Squared Error
10^2
10^1
10^0
10^-1
10^-2
10^-3
0
500
1000
1500
2000
2500
3000
Epoch



Netz: (2-)4-4-1

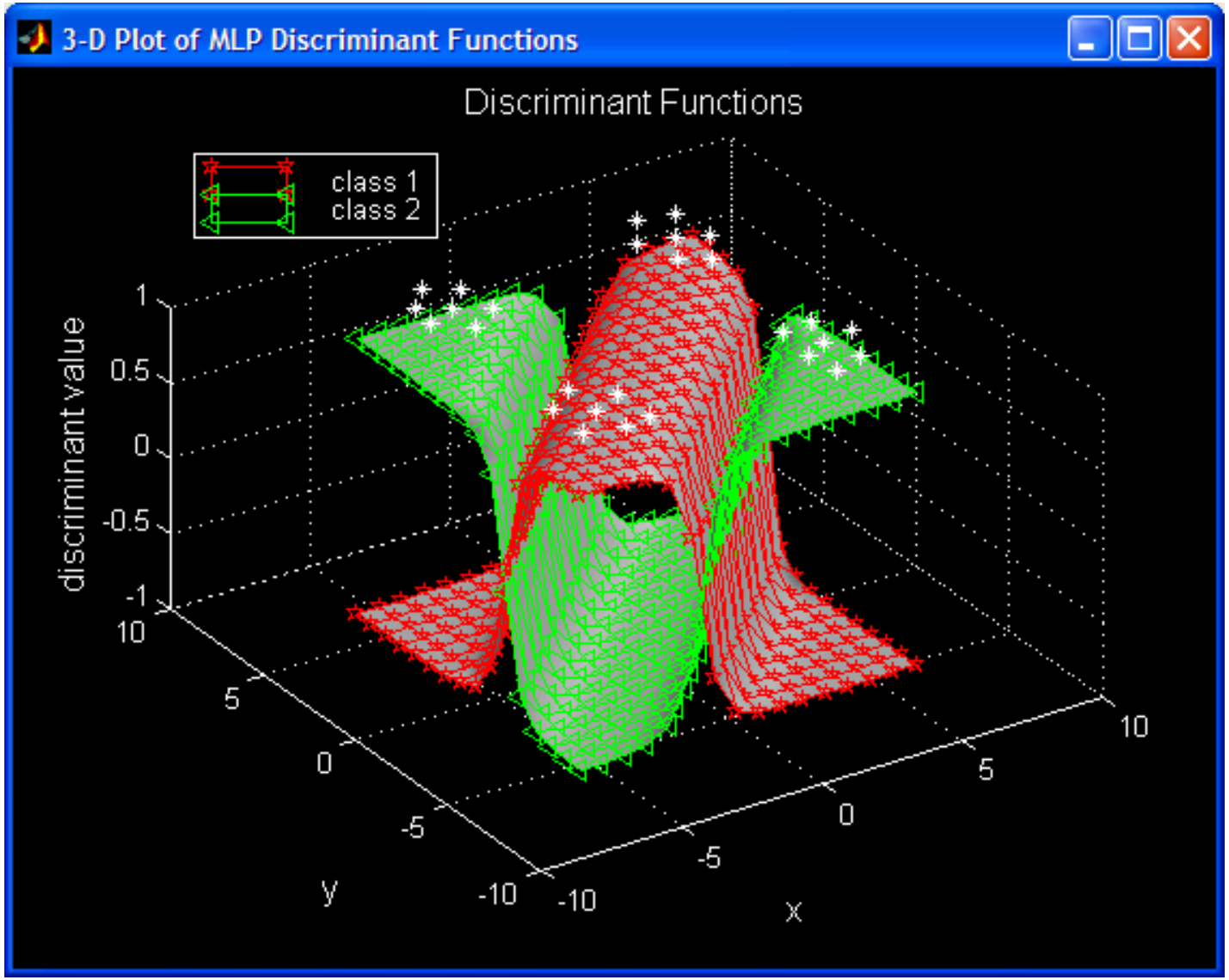

4
3.5
3
2.5
2
1.5
1
0.5
0
0
0.5
1
1.5
2
2.5
3
3.5
4

Demo mit MATLAB
(Klassifikationgui.m)

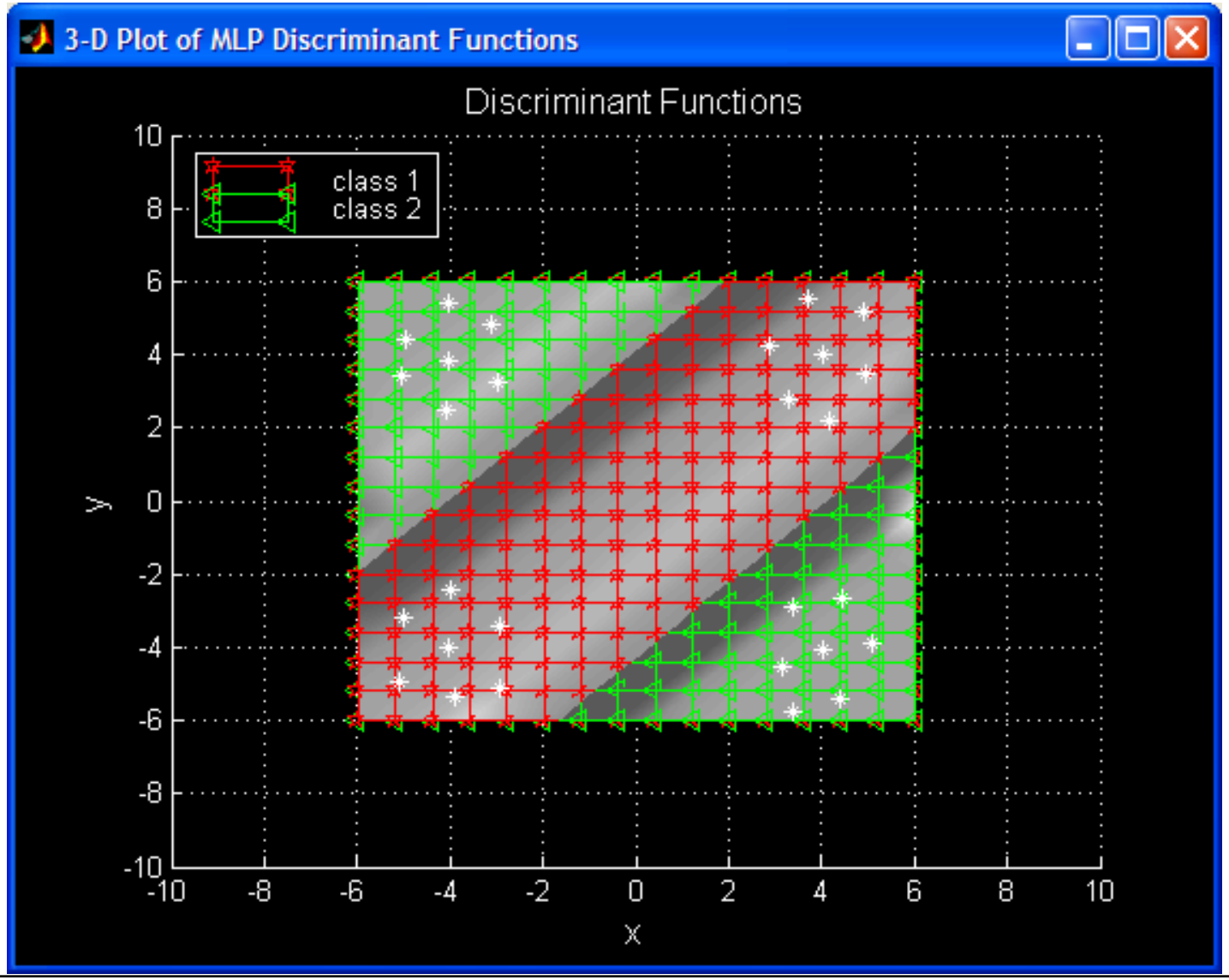
- Öffnen von Matlab
  - zuerst setmypath.m aufrufen, dann
  - C:\Home\ppt\Lehre\ME_2002\matlab\KlassifikatorEntwurf-WinXX\Klassifikationgui
  - Datensatz: Samples/xor2.mat laden
    Einstellung: 500 Epochen, Lernrate 0.7
    [2] 0.7 einfache Lösung konvergiert
    [2,2] 0.7 optimale Lösung konvergiert nicht
    [4,2] 0.7 optimale Lösung konvergiert
  - Gewichtsmatrizen laden: me-beispiele/xor-trivial.mat
  - Gewichtsmatrizen laden: me-beispiele/xor-3.mat laden
- Zweiklassenproblem mit Bananenshape eingeben
  - Datensatz Samples/banana_test_samples.mat laden
  - Voreinstellungen aus: models/MLP_7_3_banana.mat laden
    models/MLP_3_3_banana.mat laden

Start der Matlab-Demo
matlab-Klassifikation_gui.bat

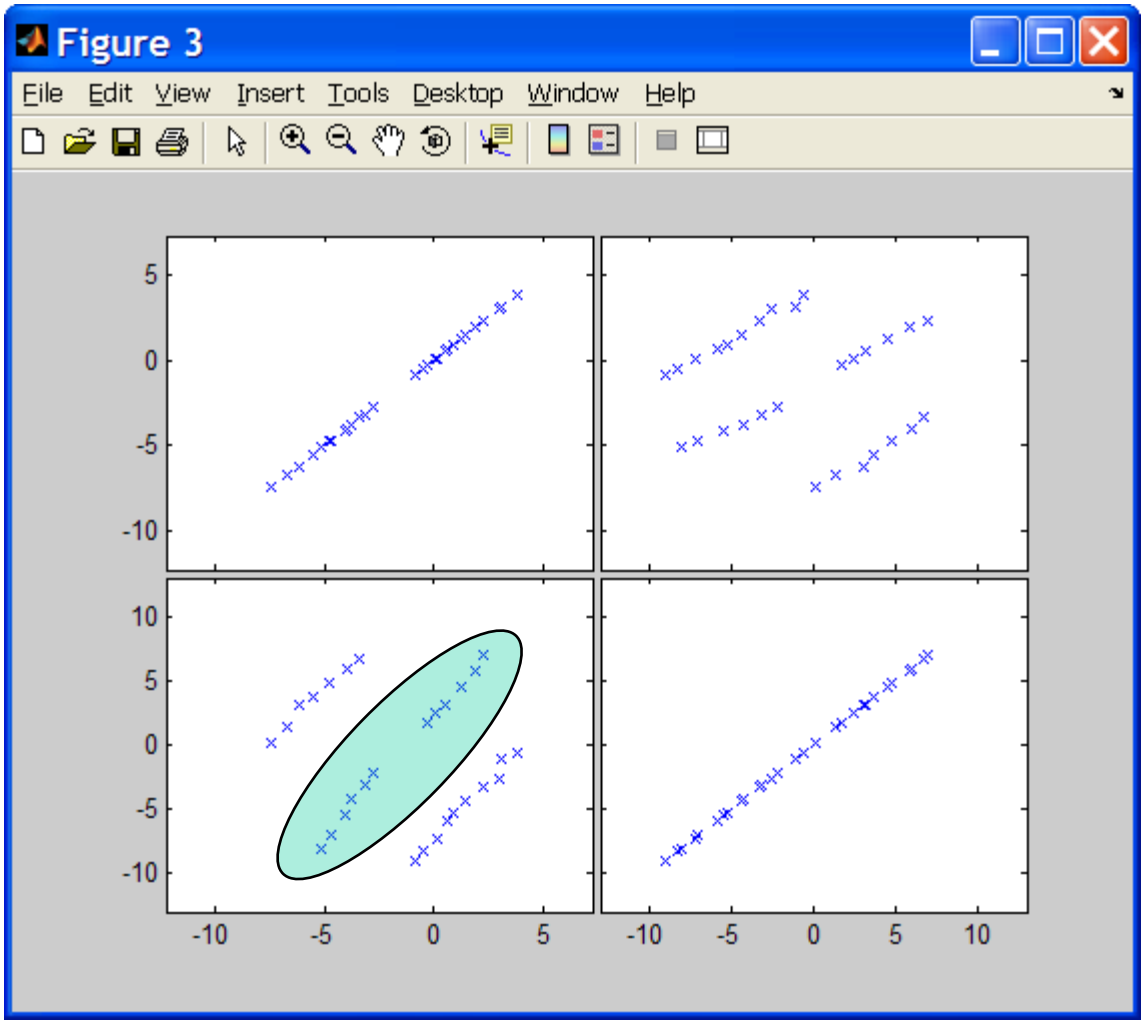
Lösung des XOR-Problems mit zweischichtigem NN



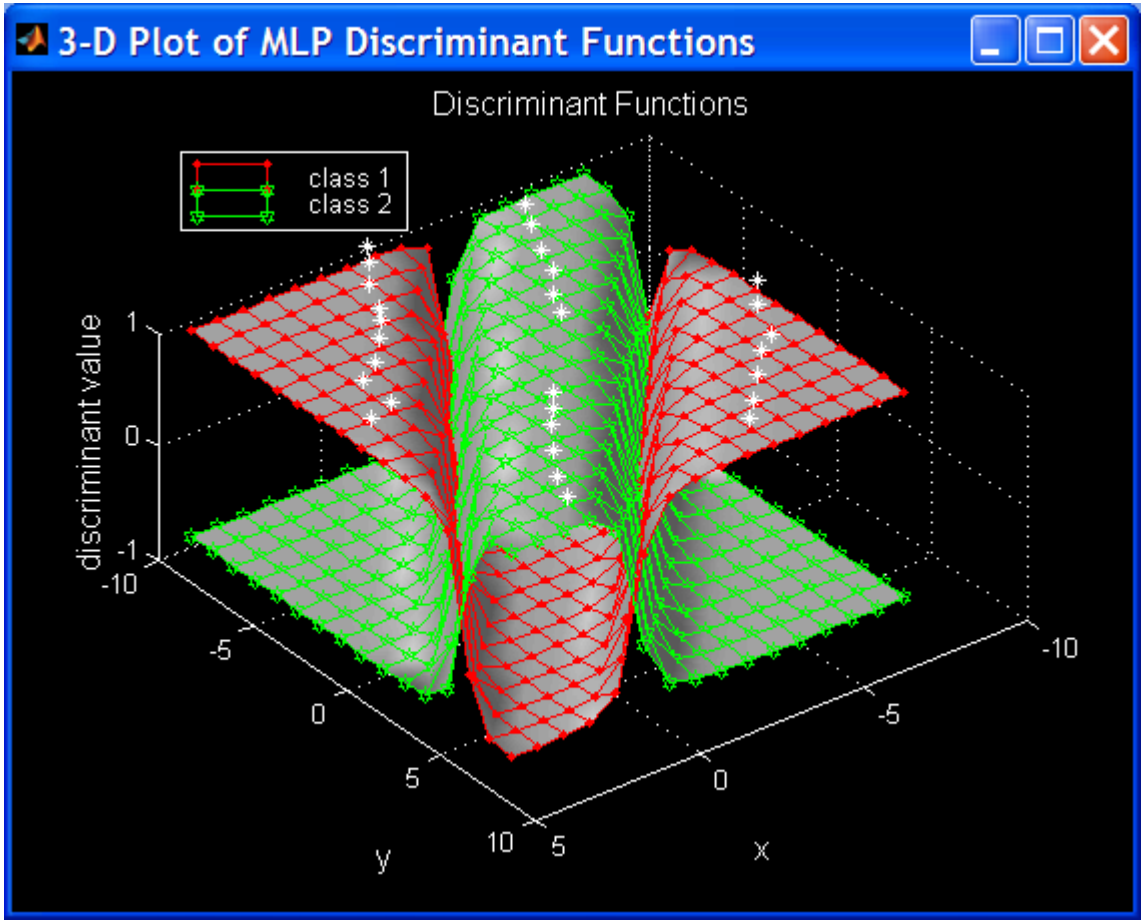
Lösung des XOR-Problems mit zweischichtigem NN



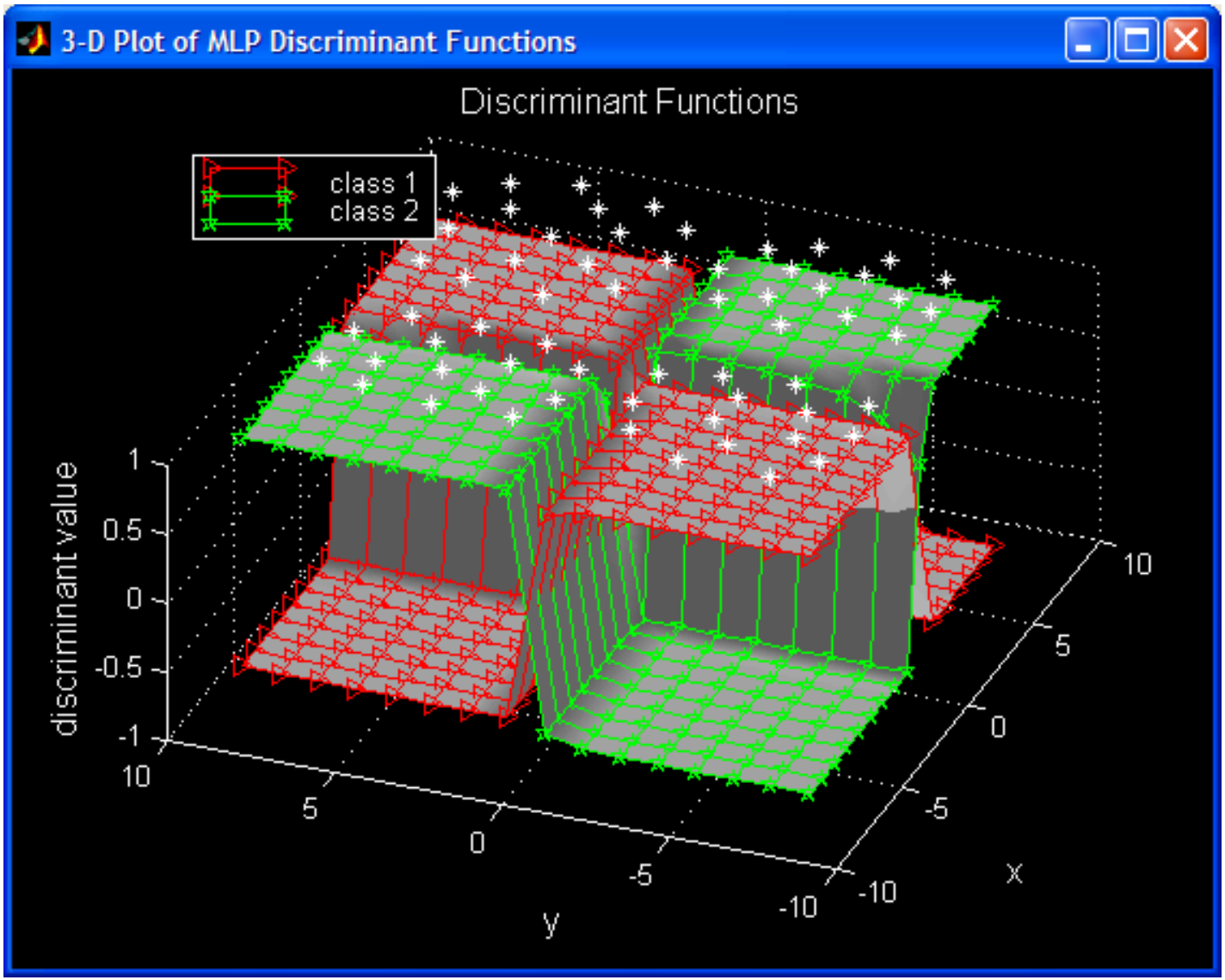
Lösung des XOR-Problems mit zweischichtigem NN



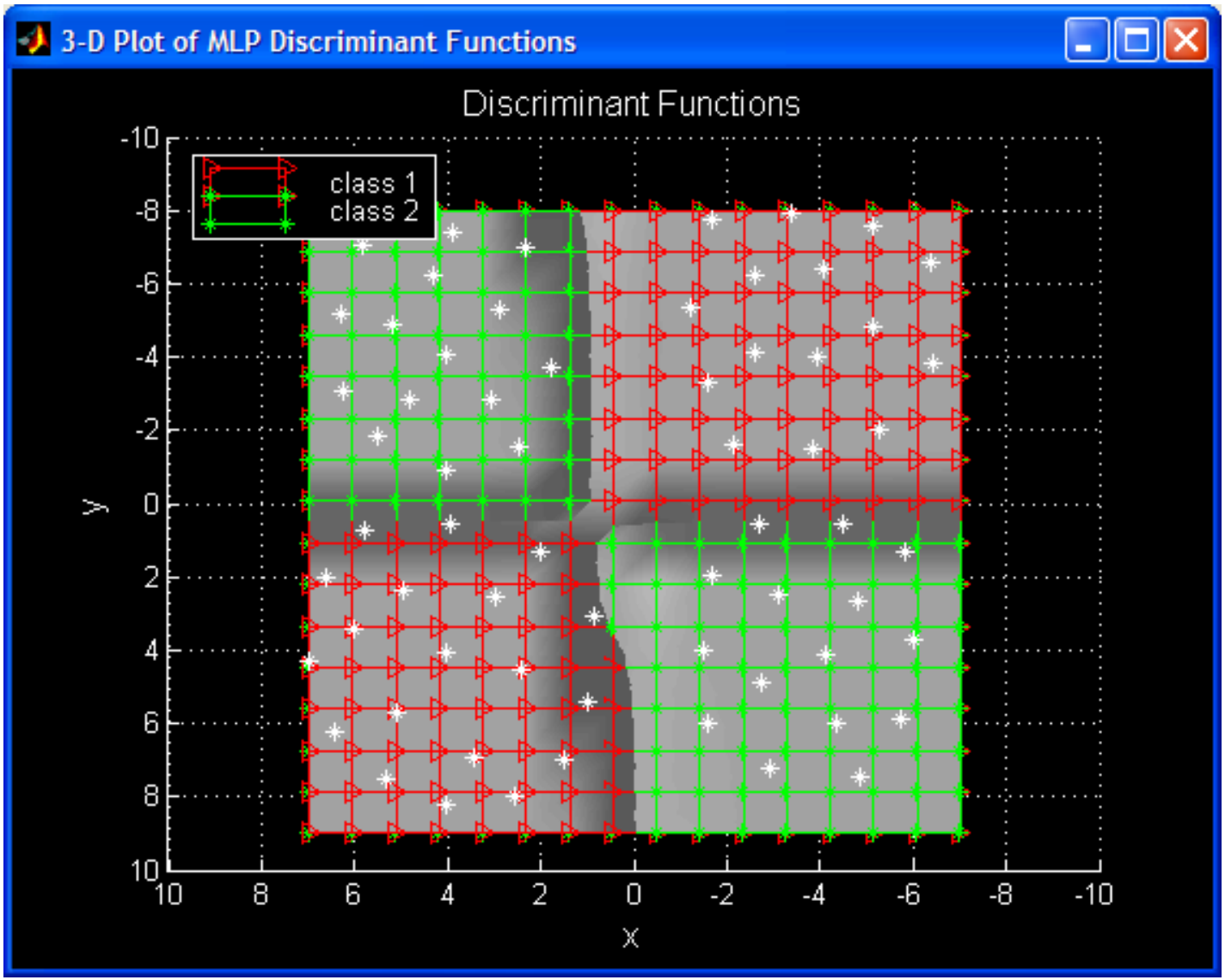
Lösung des XOR-Problems mit zweischichtigem NN



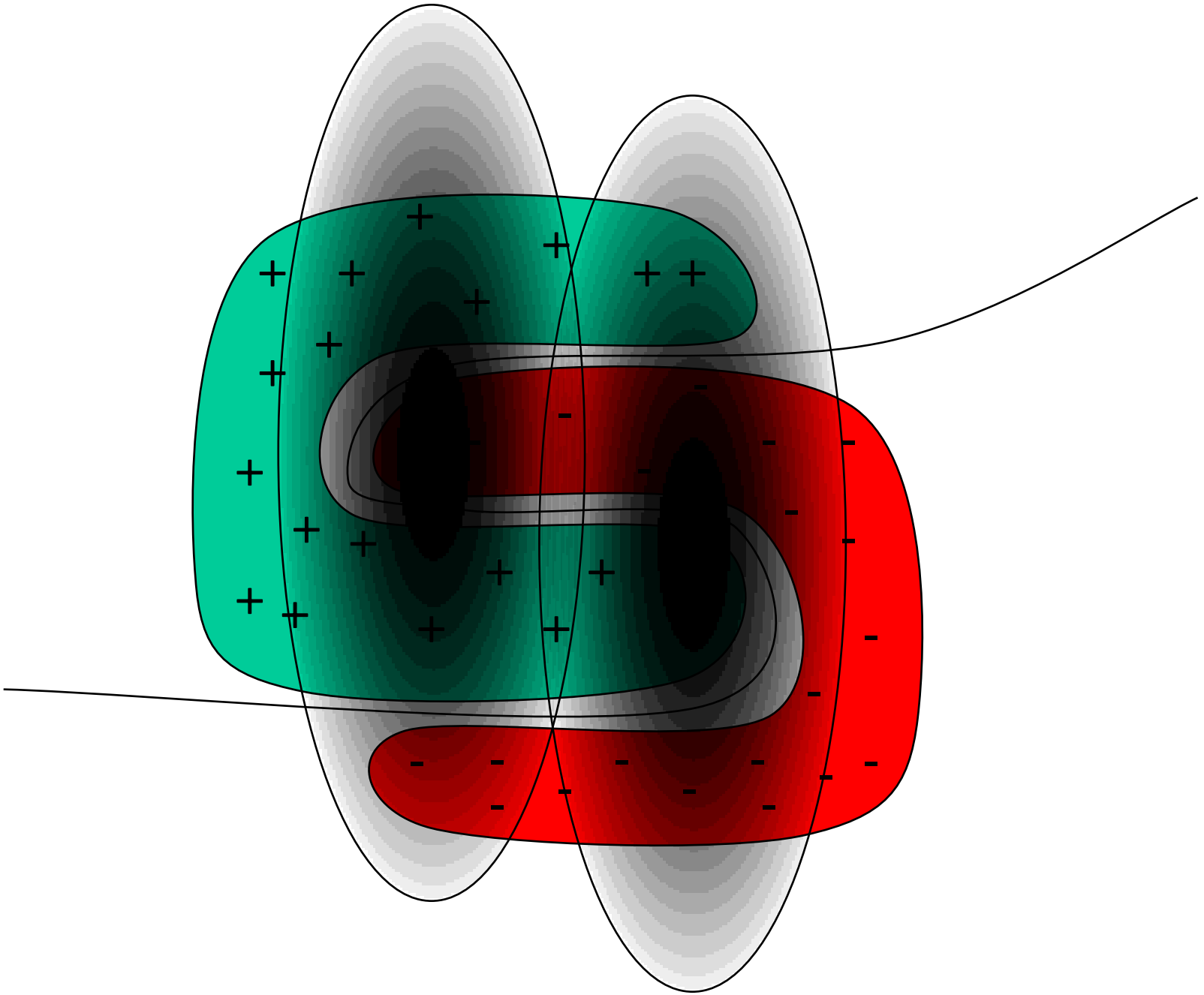
Lösung des XOR-Problems mit dreischichtigem NN

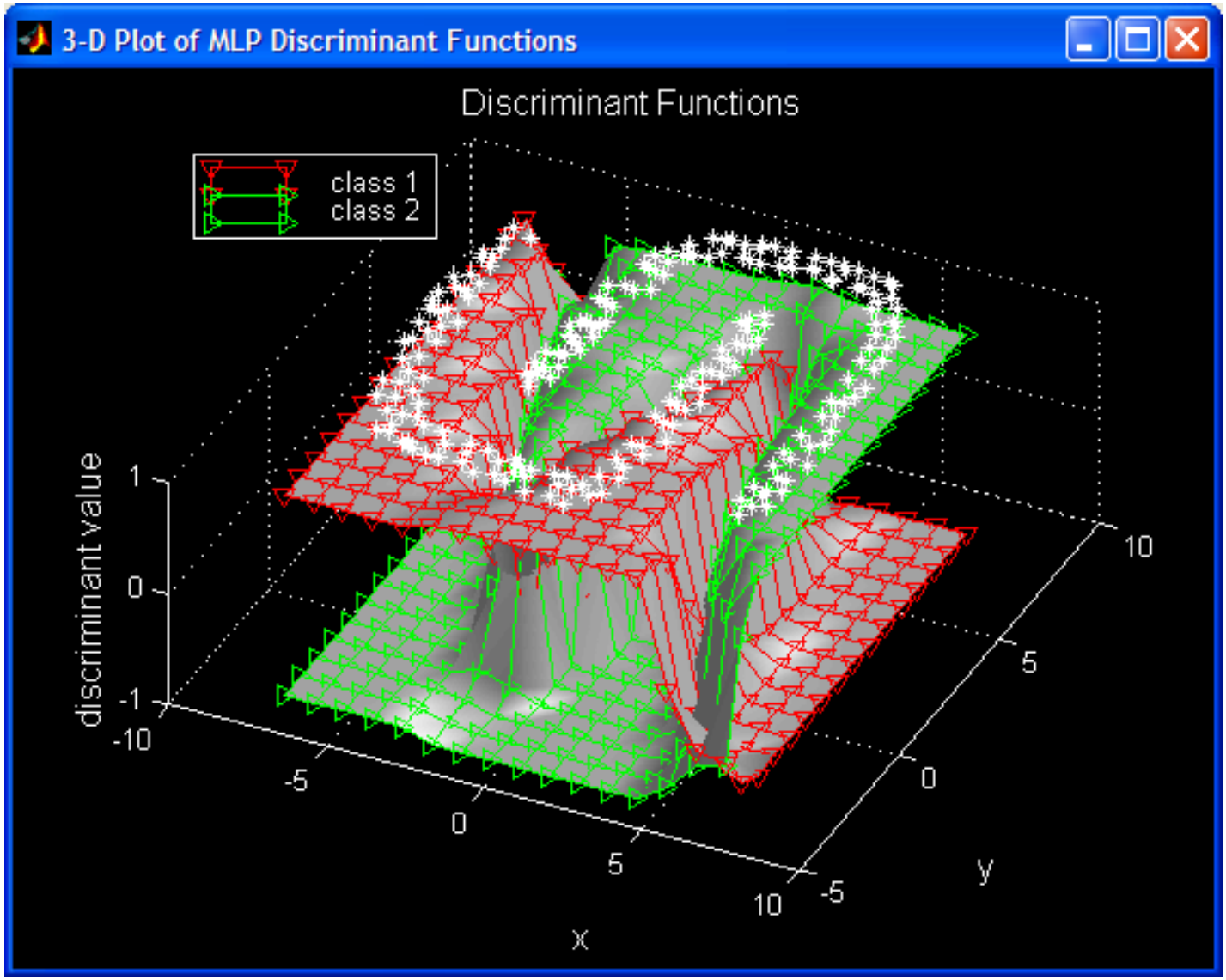


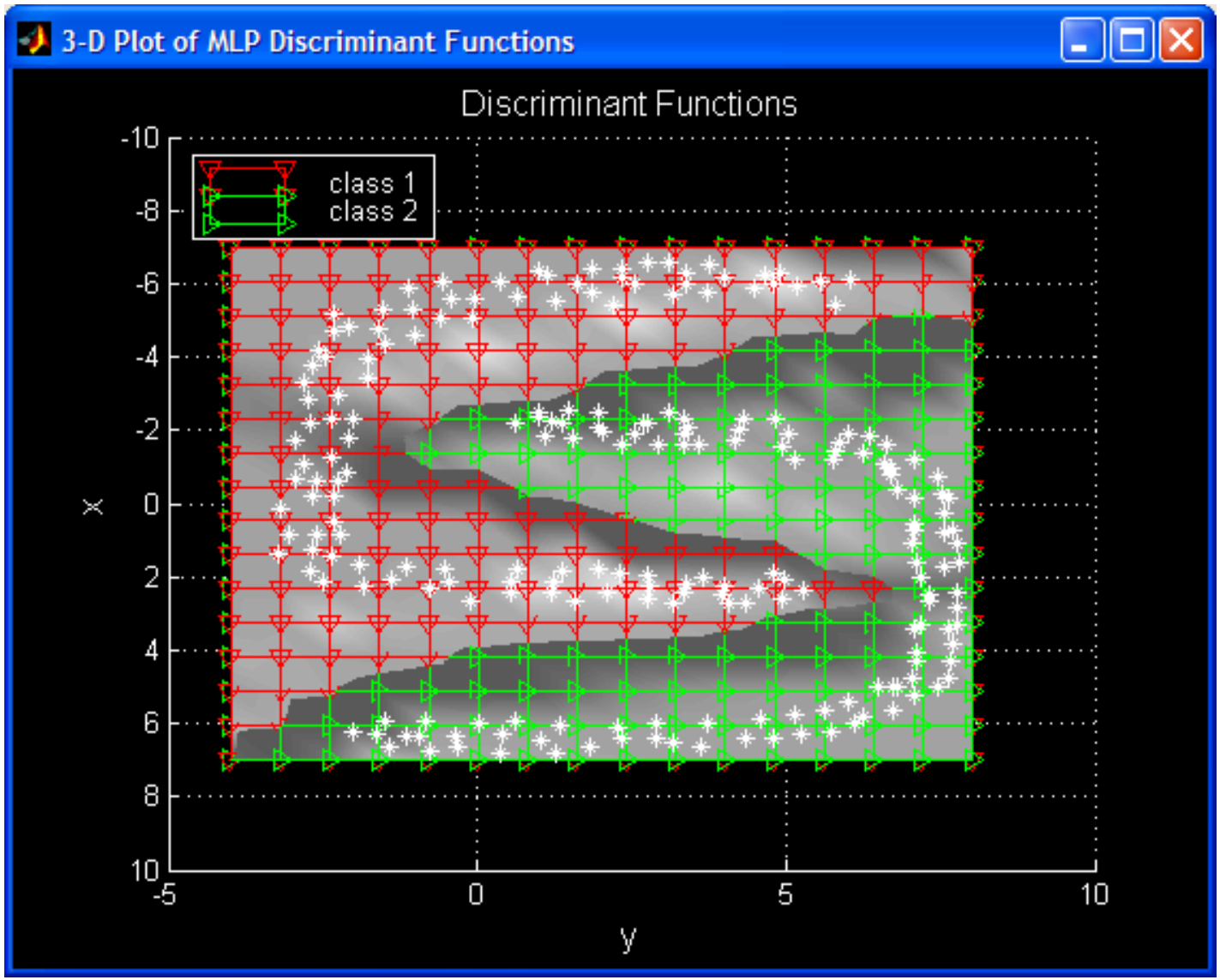
Lösung des XOR-Problems mit dreischichtigem NN



Durch Normalverteilungen schlecht darstellbare Klassen




3-D Plot of MLP Discriminant Functions
Discriminant Functions
class 1
class 2
discriminant value
1
0
-1
-10
-5
0
5
10
x
-5
0
5
10
y


3-D Plot of MLP Discriminant Functions
Discriminant Functions
class 1
class 2
-10
-8
-6
-4
-2
0
2
4
6
8
10
x
-5
0
5
10
y

Konvergenzverhalten von Backpropagation-Algorithmen

- Backpropagation mit gewöhnlichem Gradientenabstieg (steepest descent)

Start der Matlab-Demo
matlab-BP-gradient.bat

- Backpropagation mit konjugiertem Gradientenabstieg (conjugate gradient) – wesentlich bessere Konvergenz!

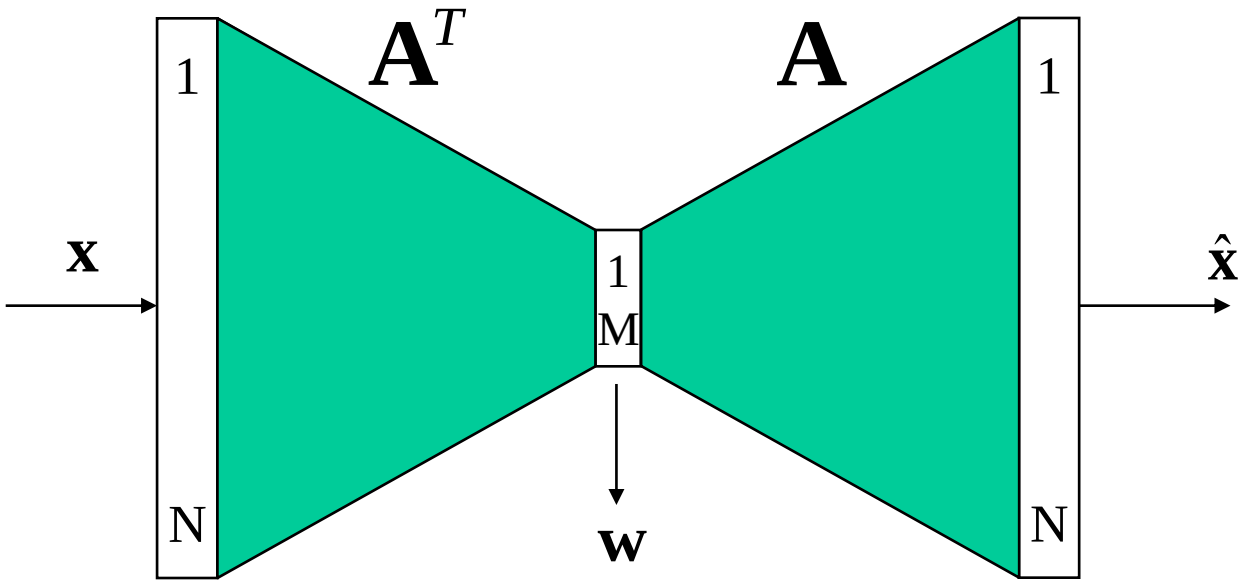
Start der Matlab-Demo
matlab-BP-CGgradient.bat

NN und deren Eigenschaften

- „Quick and Dirty“. Ein NN-Entwurf ist einfach zu realisieren und man erhält durchaus brauchbare Lösungen (eine suboptimale Lösung ist jedenfalls besser als irgendeine Lösung).
- Es können damit insbesondere auch sehr große Probleme angegangen werden.
- Alle Strategien benutzen jedoch nur „lokale“ Optimierungsstrategien und erreichen in der Regel kein globales Optimum. Dies ist der gravierende Nachteil für den Einsatz Neuronaler Netze!

Verwendung eines NN zur iterativen Berechnung der Hauptkomponentenanalyse (KLT)

- Anstatt die KLT explizit über ein Eigenwertproblem zu lösen, kann auch ein NN zur iterativen Berechnung herangezogen werden.
- Man verwendet ein zweischichtiges Perceptron. Der Ausgang der verdeckten Schicht **w** ist der gesuchte Merkmalsvektor.



- Die erste Schicht berechnet den Merkmalsvektor zu:

$$\mathbf{w} = \mathbf{A}^T\mathbf{x}$$

- Dabei wird angenommen, dass eine *lineare Aktivierungsfunktion* zum Ansatz kommt. Die zweite Schicht realisiert die Rekonstruktion von **x** mit der transponierten Gewichtsmatrix der ersten Schicht

$$\hat{\mathbf{x}} = \mathbf{A}\mathbf{w}$$

- Das Optimierungsziel ist die Minimierung von:

$$J = E\left\{\left\|\hat{\mathbf{x}} - \mathbf{x}\right\|^2\right\} = E\left\{\left\|\mathbf{A}\mathbf{w} - \mathbf{x}\right\|^2\right\} = E\left\{\left\|\mathbf{A}\mathbf{A}^T\mathbf{x} - \mathbf{x}\right\|^2\right\}$$

- Die folgende Lernregel:

$$\mathbf{A} \leftarrow \mathbf{A} - \alpha\nabla\mathbf{J} = \mathbf{A} - \alpha(\mathbf{A}\mathbf{w} - \mathbf{x})\mathbf{w}^T$$

- führt zu einer Koeffizientenmatrix **A**, welche als Produkt von zwei Matrizen geschrieben werden kann (ohne Beweis!):

$$\mathbf{A} = \mathbf{B}_M\mathbf{T}$$

- $\mathbf{B}_M$ ist eine $N \times M$-Matrix der M dominanten Eigenvektoren von $E\{\mathbf{x}\mathbf{x}^T\}$ und **T** eine orthonormale $M \times M$–Matrix, welche eine Rotation des Koordinatensystems bewirkt, innerhalb eines Raumes, welcher durch die M dominanten Eigenvektoren von $E\{\mathbf{x}\mathbf{x}^T\}$ aufgespannt wird.

- Die Lernstrategie beinhaltet keine Translation. D.h. sie berechnet die Eigenvektoren der Momentenmatrix $E\{\mathbf{x}\mathbf{x}^T\}$ und nicht der Kovarianzmatrix $\mathbf{K} = E\{(\mathbf{x}-\boldsymbol{\mu}_x)(\mathbf{x}-\boldsymbol{\mu}_x)^T\}$. Dies kann jedoch realisiert werden, indem man einen rekursiv geschätzten Erwartunghswert $\boldsymbol{\mu}_x = E\{\mathbf{x}\}$ subtrahiert, bevor man die rekursive Schätzung des Merkmalvektors beginnt. Den gleichen Effekt erzielt man dadurch, dass man einen erweiterten Beobachtungsvektor verwendet:

$$\tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix} \quad \text{anstatt} \quad \mathbf{x}$$