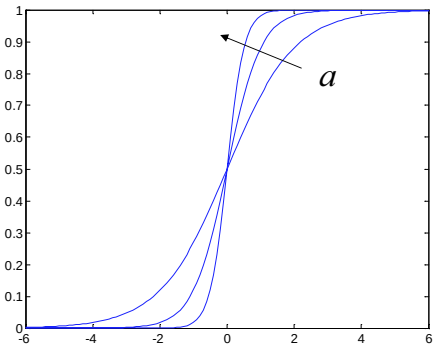
# Lernstrategien für Neuronale Netze - Der Backpropagation-Algorithmus

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

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

Sie approximiert mit wachsendem *a* die Sprungfunktion *σ(x)*.






Die Funktion *ψ(x)* hat einen engen Bezug zur tanh-Funktion. Es gilt:

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

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 $w_{i,j}$.

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

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

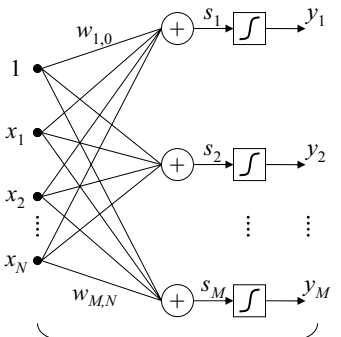
# Eine Schicht des Neuronalen Netzes

Eine einzige Schicht des NN wird charakterisiert durch die $M \times N$ Koeffizienten der Gewichtsmatrix **W**, den Offset-Vektor **b** und eine vektorielle Sigmoid-Funktion.
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.




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

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

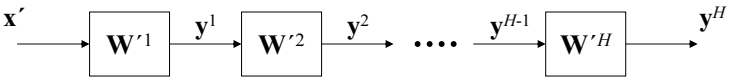


# Gestalt der erweiterten Gewichtsmatrix

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

# 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:




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

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

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

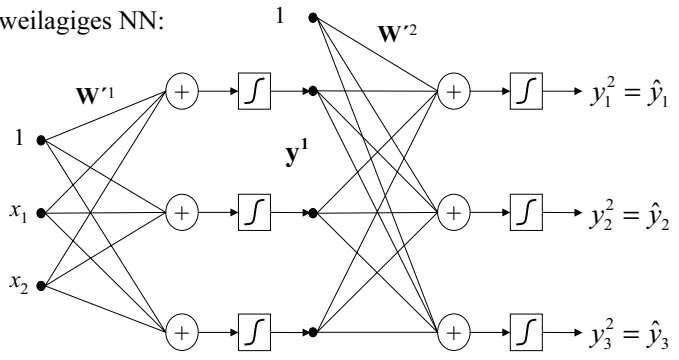
Anmerkung: Das einschichtige NN und der lineare Polynomklassifikator sind identisch, wenn die Aktivierungsfunktion $\psi \equiv 1$ gesetzt wird.



# 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:



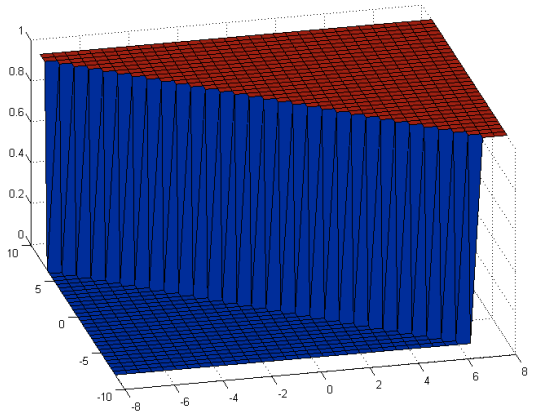
## Beziehungen zu dem Konzept der Funktionsapproximation durch eine Linearkombination von Basisfunktionen

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

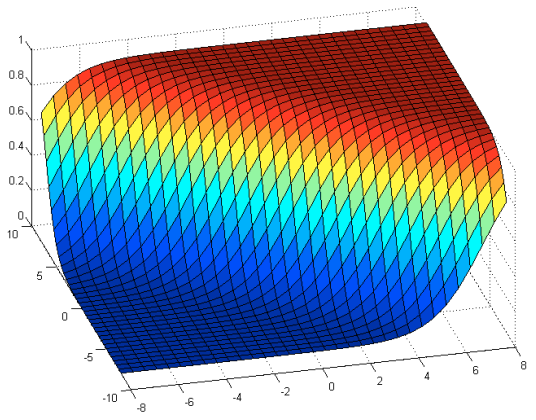
Deshalb beeinflusst die Koeffizientenmatrix $\mathbf{W}'^1$ der ersten Schicht das *Aussehen* der Basisfunktionen, während die Gewichtmatrix $\mathbf{W}'^2$ der zweiten Schicht die *Koeffizienten der Linearkombinationen* enthält. Diese wird zusätzlich durch die Aktivierungsfunktion gewichtet.



## Reaktion eines Neurons mit zwei Eingängen und einer Schwellwertfunktion σ

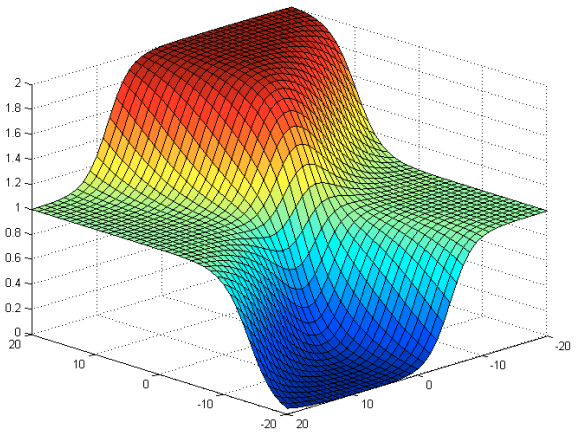


# Reaktion eines Neurons mit zwei Eingängen und einer Sigmoidfunktion ψ

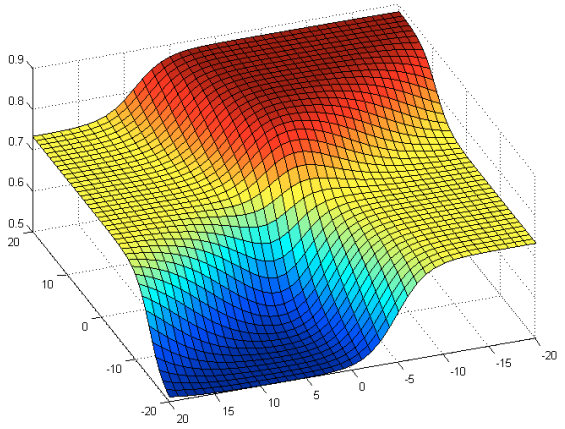





# Überlagerung zweier Neuronen mit je zwei Eingängen und einer Sigmoidfunktion



# Reaktion eines Neurons in der zweiten Schicht auf zwei Neuronen der ersten Schicht jeweils mit Sigmoidfunktionen






# Der Backpropagation-Lernalgorithmus

- Die Klassenzugehörigkeitsabbildung geschieht durch ein Multilagenperceptron, welches am 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:

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

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

- 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 nicht klar, ob ein solches existiert oder wie viel *lokale* Minima vorhanden sind.
  Der Backpropagation Algorithmus löst das Problem iterativ mit einem Gradientenalgorithmus. Iteriert wird gemäß:

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

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

- Die Iteration wird beendet, wenn der Gradient bei einem (lokalen oder auch globalen) Minimum verschwindet.
- Eine geeignete Wahl von $\alpha$ 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 (Gradienten-algorithmus).
- <u>Berechnung des Gradienten:</u>
  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.

- Da der Erwartungswert $E\{...\}$ nicht zur Verfügung steht, muss er durch den Mittelwert über einzelne Stichproben ersetzt werden:

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

- Der Gradient ergibt sich durch die Mittelung über die Stichprobengradienten:

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

- 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

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

- <u>Partielle Ableitungen für die h-te Schicht:</u>
  Für die *h*-te Schicht gilt:

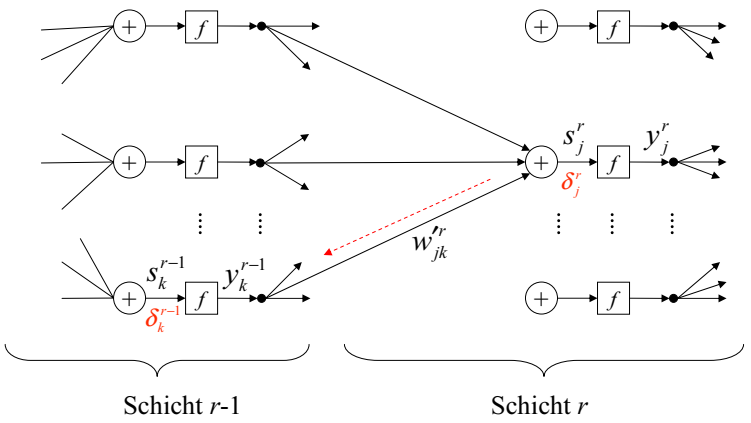
$$y_m^h = \psi(s_m^h) = \psi\left(\sum_{n=0}^{N^h} w'^h_{nm} x'^h_n\right)$$

$x'^h_n$ n-ter Eingang von Schicht *h*

$y_m^h$ m-ter Ausgang von Schicht *h*



# Definition der Variablen zweier Schichten für den Backpropagation-Algorithmus



- Wir berechnen zunächst die Wirkung 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:

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

- unter Einführung der *Empfindlichkeit* von Zelle $n$

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

- ergibt sich:

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

- und damit schließlich für das Update der Gewichte:

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



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

$$\underbrace{\frac{\partial J}{\partial s^{r-1}_{j}}}_{\delta^{r-1}_{j}} = \sum_k \underbrace{\frac{\partial J}{\partial s^{r}_{k}}}_{\delta^{r}_{k}} \frac{\partial s^{r}_{k}}{\partial s^{r-1}_{j}}$$

- Mit

$$\frac{\partial s^{r}_{k}}{\partial s^{r-1}_{j}} = \frac{\partial\left(\sum_m w'^{r}_{km} \overbrace{y^{r-1}_{m}}^{f(s^{r-1}_{m})}\right)}{\partial s^{r-1}_{j}} = w'^{r}_{kj} f'(s^{r-1}_{m})$$

- ergibt sich:

$$\delta^{r-1}_{j} = f'(s^{r-1}_{j}) \left[\sum_k w'^{r}_{kj} \delta^{r}_{k}\right]$$

- 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:

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

- Die Gradienten $\nabla\mathbf{J}$ und $\nabla\mathbf{J}_j$ 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 $[\mathbf{x}'_j, \hat{y}_j]$ sei gegeben
- Berechne die Diskriminierungsfunktion $\hat{\mathbf{y}}$ aus der gegebenen Beobachtung $\mathbf{x}'_j$ 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 $\delta_j^H$ der Ausgangsschicht und berechne daraus rückwärts alle Werte der niederen Schichten gemäß:

$$\delta_j^{r-1} = f'(s_j^{r-1})\left[\sum_k w'^r_{kj}\,\delta_m^k\right] \quad \text{für} \quad r = H, H-1, \ldots, 2$$

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

$$f'(x) = \frac{d\psi}{dx} = \psi(x)(1-\psi(x))$$

- Korrigiere (parallel) alle Perceptron-Gewichte gemäß:

$$w'^{r}_{nm} \leftarrow w'^{r}_{nm} - \alpha \frac{\partial J}{\partial w'^{r}_{nm}} = w'^{r}_{nm} - \alpha \delta^{r}_{n} y^{r-1}_{m} \quad \text{für} \quad \begin{matrix} h = 1,2,\ldots H \\ n = 1,2,\ldots N^{H} \\ m = 1,2,\ldots M^{H} \end{matrix}$$

- Beim individuellen Lernen werden die Gewichte $\{w'^{h}_{nm}\}$ mit $\nabla\mathbf{J}$ 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 $\{\nabla\mathbf{J}\}$ zu ermitteln, bevor die Gewichte korrigiert werden können gemäß:

$$w'^{r}_{nm} \leftarrow w'^{r}_{nm} - \sum_{i} \alpha \delta^{r}_{n}(i) y^{r-1}_{m}(i) \quad \text{für} \quad \begin{matrix} h = 1,2,\ldots H \\ n = 1,2,\ldots N^{H} \\ m = 1,2,\ldots M^{H} \end{matrix}$$



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 $\alpha$ 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 $(N,N^{1},\ldots, N^{H-1},K)$ zu:

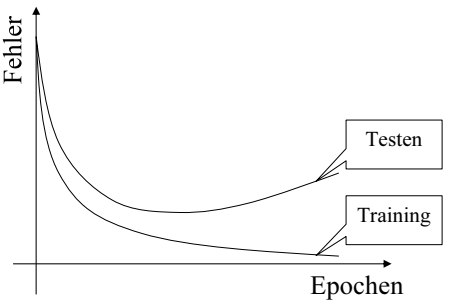
$$T = \dim(\mathbf{W}) = \sum_{h=1}^{H} (N^{h-1}+1) N^{h} \quad \text{mit } N^{0} = N \text{ und } N^{H} = K$$

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

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

# Zur Dimensionierung des Netzes

- Die Überlegungen bei dem Entwurf eines mehrschichtigen Perceptrons mit Schwellwertfunktionen ($\sigma$ 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

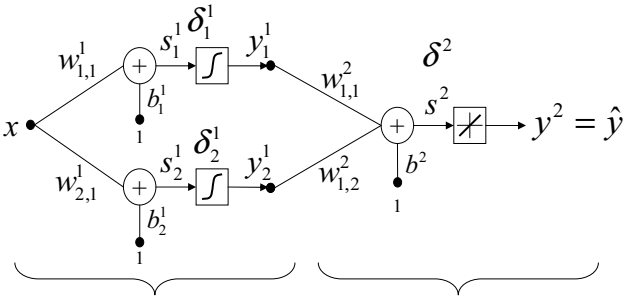
- Wenn $T$=dim(**W**) die Anzahl der Gewichte und Biasterme ist, so läßt sich einfach nachvollziehen, daß $O(T)$ Berechnungsschritte für die Vorwärtssimulation benötigt werden, $O(T)$ für das Backpropagieren des Fehlers und ebenso $O(T)$ Operationen für die Korrektur der Gewichte, also erhält man insgesamt 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:

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

- 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: $\mathbf{W}^1 = \begin{bmatrix} w^1_{1,1} \\ w^1_{2,1} \end{bmatrix}$ und $\mathbf{b}^1 = \begin{bmatrix} b^1_1 \\ b^1_2 \end{bmatrix}$ mit: $\mathbf{W}^2 = \begin{bmatrix} w^2_{1,1} & w^2_{1,2} \end{bmatrix}$

- 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 mit linearer Aktivierungsfunktion ergibt sich:

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

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

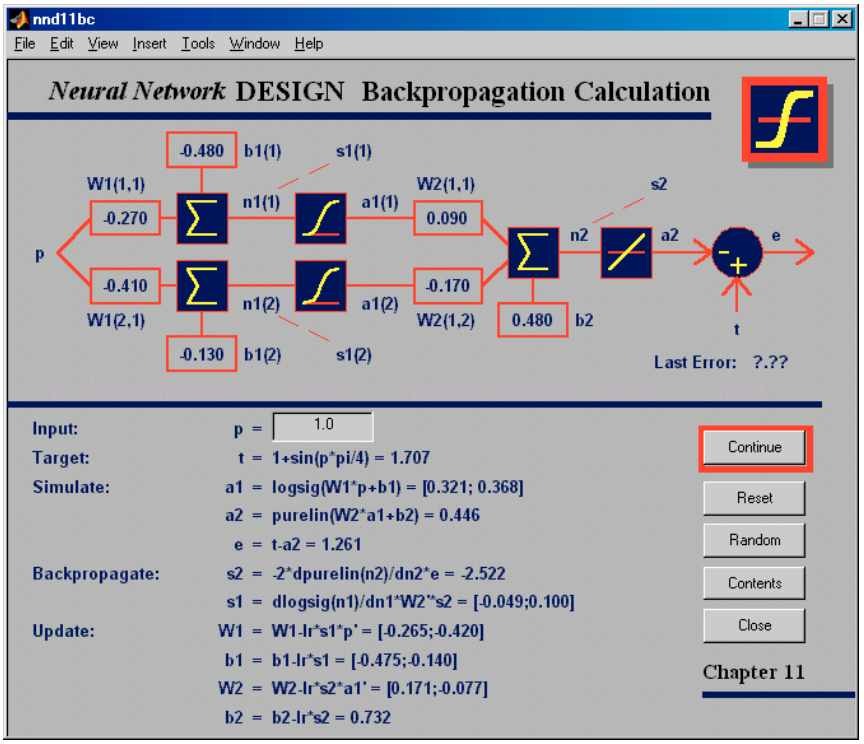
$$\delta^1_j = f'(s^1_j)\left[ w'^2_{1,j} \delta^2 \right] = s^1_j (1 - s^1_j)\left[ w^2_{1,j} \delta^2 \right] \quad \text{für} \quad j = 1,2$$

- Korrektur der Gewichte in der Ausgangsschicht:

$$\Delta w^2_{1,j} = \alpha \delta^2 y^1_j \quad \text{und} \quad \Delta b^2 = \alpha \delta^2 \quad \text{für} \quad j = 1,2$$

- Korrektur der Gewichte in der Eingangsschicht:

$$\Delta w^1_{j,1} = \alpha \delta^1_j x \quad \text{und} \quad \Delta b^1_j = \alpha \delta^1_j \quad \text{für} \quad j = 1,2$$





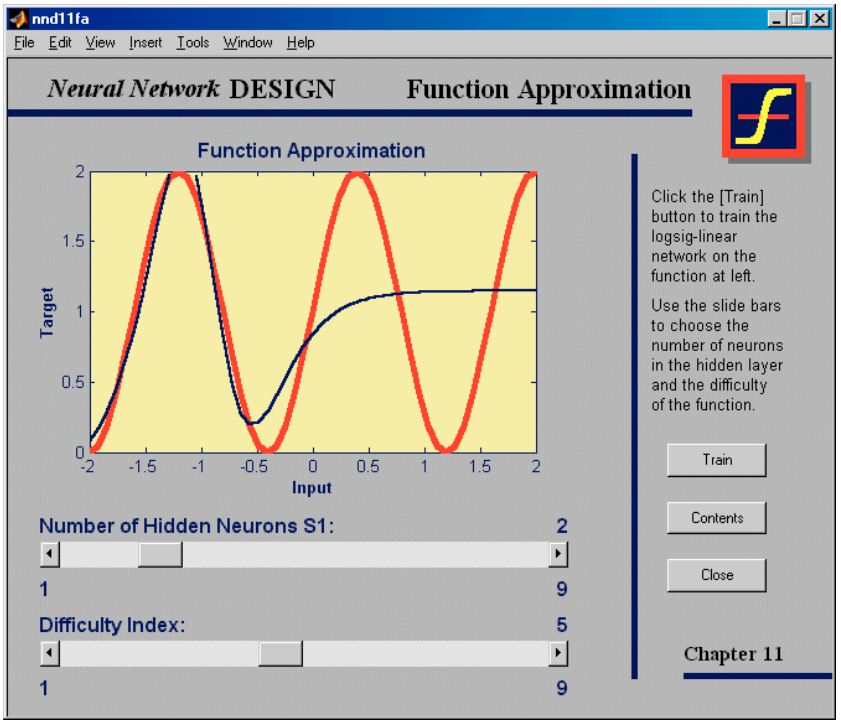

# Backpropagation zum Trainieren eines zweischichtigen Netzwerks zur Funktionsapproximation

(Matlab-Demo: „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 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.


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