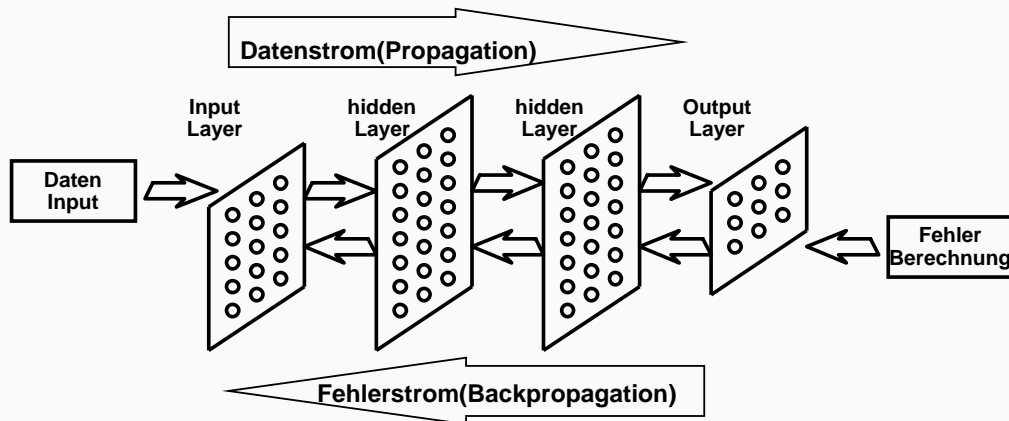


# Backpropagation

Netze ohne Rückkopplung, überwachtes Lernen, Gradientenabstieg, Delta-Regel



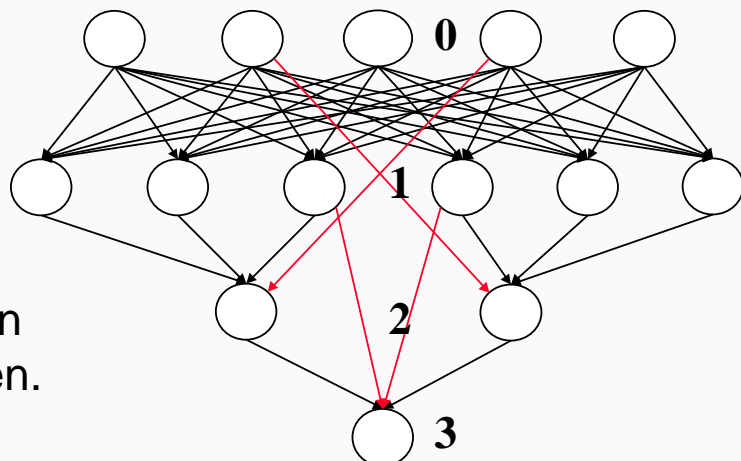
H.Werner/SS2000

Backpropagation

1

# feedforwardNetze

- Netze ohne Rückkopplungen können stets in die Form eines Schichtennetzwerkes gebracht werden, wobei aber durchaus Verbindungen eine oder mehrere Schichten überspringend dürfen.
- $a \rightarrow b$  (es gibt einen Weg von  $a$  nach  $b$ ) ist reflexiv und antisymmetrisch, also können die Schichten z.B. nach dem längsten Weg aufgebaut werden.



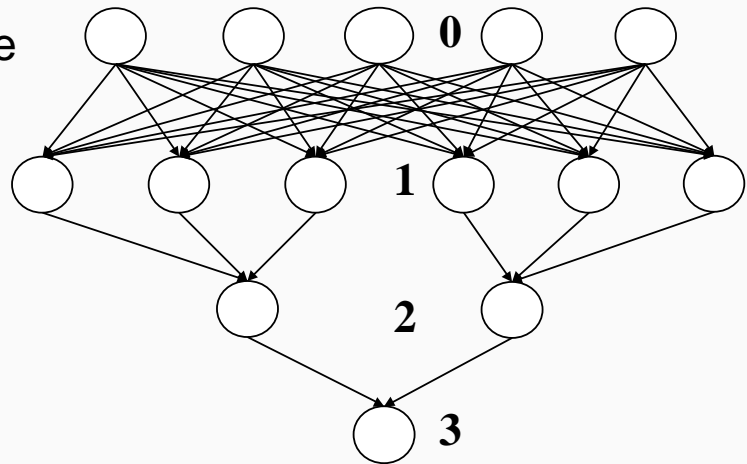
H.Werner/SS2000

Backpropagation

2

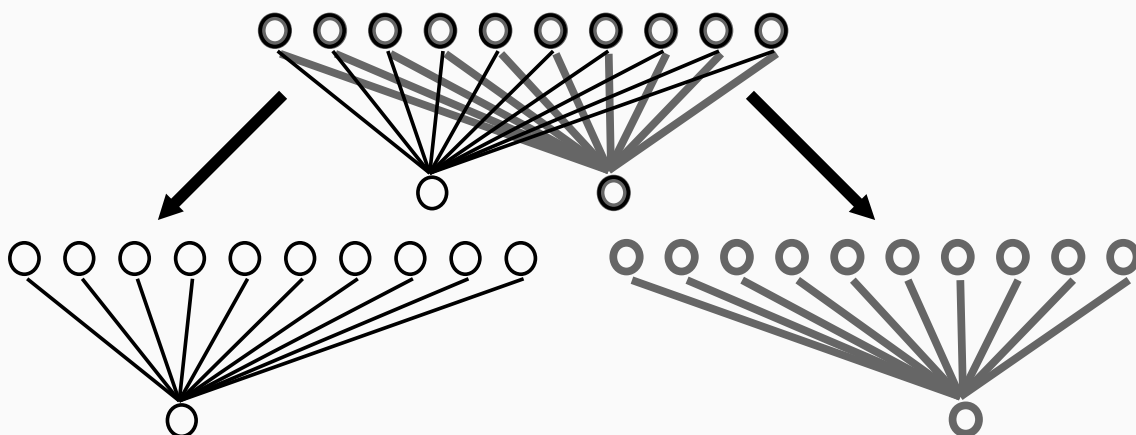
# strengeSchichten-Netze

- Wenn in einem Schichtenetz keine Verbindungen existieren, die eine oder mehrere Schichten überspringen, spricht man von einem **strengen** Schichtenetz.
- In diesem Fall ist jede Schicht durch die Länge des Weges zu ihren Neuronen gekennzeichnet.
- Die Input-Schicht hat Weglänge 0.
- Die Output-Schicht hat maximale Weglänge.



# einfacheSchicht

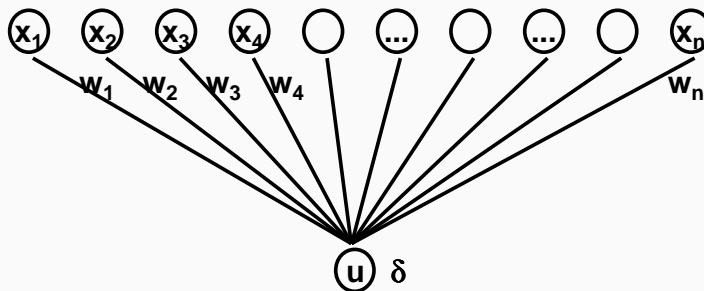
- Die beiden Neuronen der 2. Schicht beeinflussen einander nicht, deshalb können sie voneinander getrennt betrachtet werden.



- Bei mehrschichtigen Netzen geht die Unabhängigkeit verloren, d.h. sie können nicht so getrennt werden.

## $\Delta \mathbf{w}_i := \alpha \cdot \mathbf{x}_i \cdot \mathbf{e}$ (Synapsenveränderung)

- Schwellwert:  $\delta$
- Netz-Output:  $u = \theta(x_1 w_1 + x_2 w_2 + x_3 w_3 + \dots + x_n w_n - \delta)$
- erwarteter Output:  $t$  (target)
- gemachter Fehler:  $e = t - u$  (error)
- Lernkonstante:  $\eta$

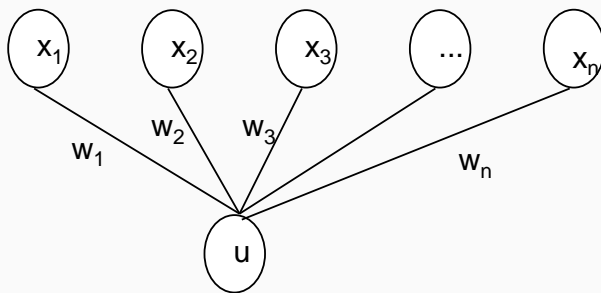


## $\Delta$ -Regel( vektoriell)

- Aktivität der Input-Schicht:  
 $\underline{\mathbf{x}} = (x_1, x_2, x_3, \dots, x_n, 1)$
- Gewichtsvektor (einschl. Schwellwert):  
 $\underline{\mathbf{w}} = (w_1, w_2, w_3, \dots, w_n, -\delta)$
- Aktivität des Ausgabeneurons:  
 $y = \theta(\underline{\mathbf{x}} \bullet \underline{\mathbf{w}})$
- Fehler des Ausgabeneurons:  
 $e = t - y$
- Gewichtsänderung:  
 $\Delta \underline{\mathbf{w}} := \alpha \cdot \mathbf{e} \cdot \underline{\mathbf{x}}$

# $\Delta$ -Regel als Ableitung

## • Perceptron:



$$u = \theta(\underline{w} \cdot \underline{x})$$

## • $\Delta$ -Regel:

$$\Delta w_i = \alpha \cdot (u - t) \cdot x_i$$

## • Fehlergradient:

$$F = (u - t)^2 = (\theta(\underline{w} \cdot \underline{x}) - t)^2$$

$$\begin{aligned} \frac{\partial F}{\partial w_i} &= \frac{\partial (u - t)^2}{\partial w_i} \\ &= \frac{\partial (\theta(\underline{w} \cdot \underline{x}) - t)^2}{\partial w_i} \\ &= 2 \cdot (u - t) \cdot \theta'(\underline{w} \cdot \underline{x}) \cdot x_i \end{aligned}$$

- Die Delta-Regel kann also interpretiert werden als der Gradientenabstieg mit dem (variablen) Lernfaktor

$$\alpha = 2 \cdot \theta'(\underline{w} \cdot \underline{x})$$

# 2-layer-Perceptron

## • 2-Layer Perceptron

Input-Vektor  $\underline{x}$

Gewichtsmatrix  $\underline{V}$

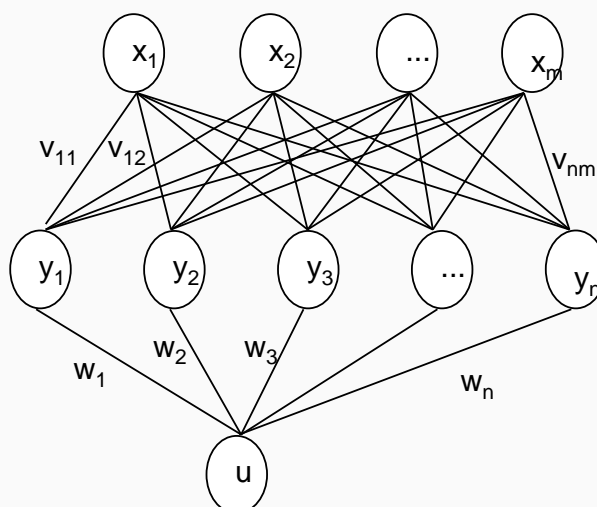
Aktivitätsvektor  $\underline{y}$

Gewichtsvektor  $\underline{w}$

Output

$$\underline{y} = \theta(\underline{V} \cdot \underline{x})$$

$$u = \theta(\underline{w} \cdot \underline{y})$$



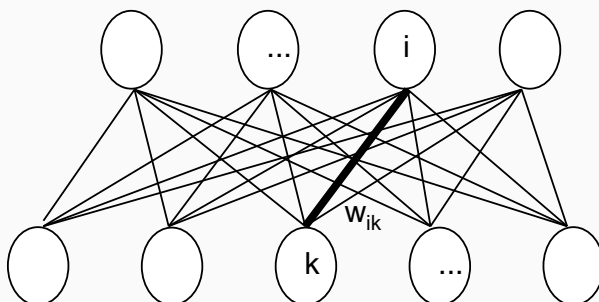
Propagierung

## 2-Layer-Fehler-Gradient

- $F = (u - t)^2 = (\theta(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) - t)^2$
- $\frac{\partial F}{\partial w_i} = \frac{\partial (\theta(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) - t)^2}{\partial w_i} = \frac{\partial (u - t)^2}{\partial w_i}$   
 $= 2 \cdot (u - t) \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) \cdot \frac{\partial (\underline{\mathbf{w}} \cdot \underline{\mathbf{y}})}{\partial w_i}$   
 $= 2 \cdot (u - t) \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) y_i$
- $\frac{\partial F}{\partial v_{ij}} = \frac{\partial F}{\partial y_i} \cdot \frac{\partial y_i}{\partial v_{ij}}$   
 $= \frac{\partial F}{\partial y_i} \cdot \frac{\partial \theta(\underline{\mathbf{v}}_i \cdot \underline{\mathbf{x}})}{\partial v_{ij}} \quad (\text{FehlervonNeuron } i)$   
 $= \frac{\partial F}{\partial y_i} \cdot \theta'(\underline{\mathbf{v}}_i \cdot \underline{\mathbf{x}}) \cdot x_j$   
 $= \frac{\partial F}{\partial u} \cdot \frac{\partial u}{\partial y_i} \cdot \theta'(\underline{\mathbf{v}}_i \cdot \underline{\mathbf{x}}) \cdot x_j$   
 $= \frac{\partial F}{\partial u} \cdot \frac{\partial \theta(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}})}{\partial y_i} \cdot \theta'(\underline{\mathbf{v}}_i \cdot \underline{\mathbf{x}}) \cdot x_j$   
 $= \frac{\partial F}{\partial u} \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) \cdot w_i \cdot \theta'(\underline{\mathbf{v}}_i \cdot \underline{\mathbf{x}}) \cdot x_j$   
 $= 2 \cdot (u - t) \cdot \theta'(\underline{\mathbf{w}} \cdot \underline{\mathbf{y}}) \cdot w_i \cdot \theta'(\underline{\mathbf{v}}_i \cdot \underline{\mathbf{x}}) \cdot x_j$

## Multi-Layer-Perceptron(MLP)

- Wie bei 2 Layern können wir auch bei mehr Layern den Gradienten schichtweise zurückberechnen, dazu interpretieren wir den Output-Fehler von Neuron j als  $e_j = \frac{\partial F}{\partial x_j}$   
d.h. für ein Output-Neuron j:  $e_j = 2(u_j - t_j)$
- Wir betrachten nun zwei aufeinanderfolgende Layer



# MLPFehlergradient

- Die Aktivität von Neuron  $j$  sei  $a_j$  und sein Output  $x_j = \theta(a_j)$
- $w_{ik}$  bezeichnet die Verbindung von Neuron  $i$  zu Neuron  $k$
- für ein verborgenes Neuron gilt dann die Rekursion:
- $$e_i = \frac{\partial F}{\partial x_i}$$

$$= \sum_k \frac{\partial F}{\partial x_k} \cdot \frac{\partial x_k}{\partial x_i} \quad (\text{alle Neuronen aus der nächsten Schicht})$$

$$= \sum_k e_k \cdot \frac{\partial \theta(\sum_j w_{jk} \cdot x_j)}{\partial x_i}$$
- $$e_i = \sum_k w_{ik} \cdot e_k \cdot \theta'(a_k).$$
- $$\frac{\partial F}{\partial w_{ik}} = \frac{\partial F}{\partial x_k} \cdot \frac{\partial x_k}{\partial w_{ik}}$$

$$= e_k \cdot \frac{\partial \theta(\sum_j w_{jk} \cdot x_j)}{\partial w_{ik}}$$
- $$\Delta w_{ik} = \alpha \cdot e_k \cdot \theta'(a_k) \cdot x_i.$$

# Backpropagation Algorithmus

- $f_j$  bezeichnet den "echten" Fehler  $\theta'(a_j)$  von Neuron  $j$ ,
- Rekursionsanfang:  
für ein Output-Neuron  $j$  sei  $f_j = 2(u_j - t_j) \cdot \theta'(a_j).$
- für ein verborgenes Neuron gilt dann die Rekursion:
- $$f_j = e_j \cdot \theta'(a_j) = \sum_k w_{jk} \cdot e_k \cdot \theta'(a_k) \cdot \theta'(a_j)$$

$$f_j = \theta'(a_j) \cdot \sum_k w_{jk} \cdot f_k$$
(Rückpropagierung des Fehlers).
- Gewichtsanpassung
$$\Delta w_{ik} = \alpha \cdot e_k \cdot \theta'(a_k) \cdot x_i$$

$$\Delta w_{ik} = \alpha \cdot f_k \cdot x_i.$$

# Outputfunktionen

- BeliebtsindbeimBackpropagatingFunktionen  $\theta$ , bei denensich  $\theta'(x)$ durch  $\theta'(x)$ ausdrückenläßt,weildann dieAktivitätdesNeuronszugunstenseinesOutputsbeider Fehlerberechnungeliminiertwerdenkann.
- Sigmoidfunktion  
 $y=1/(1+e^{-sx})$ hatdieAbleitung $y'=s \cdot (y-y^2)=s \cdot y \cdot (1-y)$
- Tangenshyperbolicus  
 $y=\tanh x$ hatalsAbleitung $y'=1-y^2$
- Glockenkurve  
 $y=1/(1+s \cdot x^2)$ hatdieAbleitung $y'=-2 \cdot y^2 \cdot \sqrt{s \cdot (y-1)}$
- Gaußkurve  
 $y=e^{-sx^2}$ hatdieAbleitung $y'=-2 \cdot y \cdot \sqrt{-s \cdot \log y}$

# Neuronenoutputs

- WirbetrachtendieVektoren  $\mathbf{a}^t=(a_1,...,a_k)$ aller  $\mathbf{x}^t=(x_1,...,x_k)$  allerNeuronenaktivitätenund-outputsüberalleSchichten genommen.
- $\mathbf{W}^1,...,\mathbf{W}^n$  seiendieMatrizenderVerbindungsgewichte von deri-1-tenzuri-tenSchicht ( $i=1,...,n$ )
- Dann giltfürdieGewichtsmatrix  $\mathbf{W}$ desgesamtenNetzes dieGleichung  $\theta(\mathbf{W} \cdot \mathbf{x})=\mathbf{a}$ ,und  $\theta(\mathbf{a})=\mathbf{x}$ d.h. :

$$\begin{pmatrix} \mathbf{E} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{W}^1 & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{W}^2 & \dots & \mathbf{0} & \mathbf{0} \\ \vdots & \vdots & & \vdots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{W}^n \dots & \mathbf{0} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_k \end{pmatrix} = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_k \end{pmatrix}$$

# Neuronenfehler

- Wir betrachten auch die Fehlervektoren  $\mathbf{e}^t = (e_1, \dots, e_k)$  und  $\mathbf{f}^t = (f_1, \dots, f_k)$  über alle Neuronen und die Gradientenmatrix  $\mathbf{D}$  von  $\theta(\mathbf{a})$ , eine Diagonalmatrix mit den Ableitungen  $\theta'(a_i)$  auf der Diagonale, d.h.  $\mathbf{f} = \mathbf{D} \cdot \mathbf{e}$ .
- Die Fehlerbackpropagation besagt dann gerade:
- $f_j := 2 \cdot (t_j - x_j) \cdot \theta'(a_j)$  für alle Output-Neuronen  $j$
- $\mathbf{f} = \mathbf{D} \cdot \mathbf{V} \cdot \mathbf{f}$ ,  $\mathbf{e} = \mathbf{V} \cdot \mathbf{f} = \mathbf{V} \cdot \mathbf{D} \cdot \mathbf{e}$  mit

$$\mathbf{V} = \begin{pmatrix} 0 & \mathbf{W}^{1t} & \dots & 0 & 0 \\ \vdots & \vdots & \dots & \vdots & \vdots \\ 0 & 0 & \dots & \mathbf{W}^{n-1t} & 0 \\ 0 & 0 & & 0 & \mathbf{W}^{nt} \\ 0 & 0 & \dots & 0 & \mathbf{E} \end{pmatrix}$$

# BP-Interpretation

- Wir können mit diesen Schreibweisen Propagierung und den Backpropagation Algorithmus wie folgt interpretieren:
- Ein Anfangsvektor  $\mathbf{x}$ , der mit der Eingabe beginnt, wird so lang mit  $\mathbf{W}$  immer wieder multipliziert und in  $\theta$  eingesetzt, bis er sich nicht mehr ändert.
- Nun wird ein Zielvektor  $\mathbf{t}$ , der auf dem Target output endet, von  $\mathbf{x}$  subtrahiert und der so entstandene Fehlervektor  $\mathbf{e}$  so lang mit der Matrix  $\mathbf{V} \cdot \mathbf{D}$  (V ist fast die Transponierte von  $\mathbf{W}$ ) multipliziert, bis er sich nicht mehr ändert.
- Schließlich wird  $\mathbf{W}$  um den Summand  $\Delta \mathbf{W} = \alpha \cdot \mathbf{x} \cdot \mathbf{D} \cdot \mathbf{e}^t$  abgewandelt.



# BP-Ableitung

- Betrachtet man nun die Fehlerals Funktion von  $\mathbf{W}$ , so sieht man:

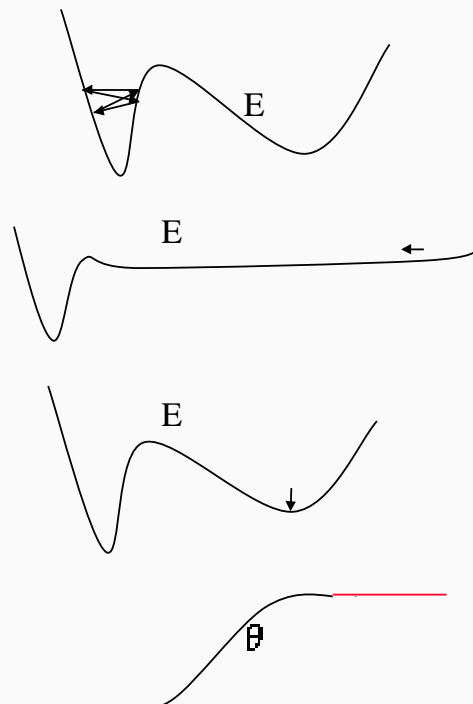
- $e(\mathbf{W})^2 = (\mathbf{t} - \mathbf{x})^2 = (\mathbf{t} - \theta(\mathbf{W} \cdot \mathbf{x}))^2$

- $$\begin{aligned}
 e(\mathbf{W} + \Delta \mathbf{W})^2 &= (\mathbf{t} - \theta((\mathbf{W} + \Delta \mathbf{W}) \cdot \mathbf{x}))^2 \\
 &= (\mathbf{t} - \theta(\mathbf{W} \cdot \mathbf{x} + \Delta \mathbf{W} \cdot \mathbf{x}))^2 \\
 &\approx (\mathbf{t} - \theta(\mathbf{W} \cdot \mathbf{x}) - \mathbf{D} \Delta \mathbf{W} \cdot \mathbf{x})^2 \quad // \text{D ist der Gradient von } \theta \\
 &\approx (\mathbf{t} - \theta(\mathbf{W} \cdot \mathbf{x}))^2 - 2 \cdot (\mathbf{D} \Delta \mathbf{W} \cdot \mathbf{x})^t \cdot (\mathbf{t} - \theta(\mathbf{W} \cdot \mathbf{x})) \\
 &= e^2 - 2 \cdot \mathbf{x}^t \Delta \mathbf{W} \cdot \mathbf{D} \cdot \mathbf{e} \quad // \Delta \mathbf{W} = \alpha \cdot \mathbf{x} \cdot \mathbf{D} \cdot \mathbf{e}^t = \alpha_{/2} \cdot \mathbf{x} \cdot \mathbf{f}^t \\
 &= e^2 - \alpha \cdot \mathbf{x}^2 \cdot \mathbf{f}^t \cdot \mathbf{D} \cdot \mathbf{e} \\
 &= e^2 - \alpha_{/2} \cdot \mathbf{x}^2 \cdot \mathbf{f}^2 \\
 &< e^2 \quad \text{für } \alpha > 0 \text{ und } \mathbf{f} \neq 0.
 \end{aligned}$$

- Beachte, daß für diese Ableitung der Aufbau von  $\mathbf{W}$  keine Rolle spielt.

# Probleme des BP

- Oszillation in engen Schluchten
- Stagnation auf flachen Plateaus
- lokale Minima
- Flat Spots ( $\theta'(a_j) \approx 0$ )  
kaum Veränderung im Training



# VariantendesBP-Algorithmus

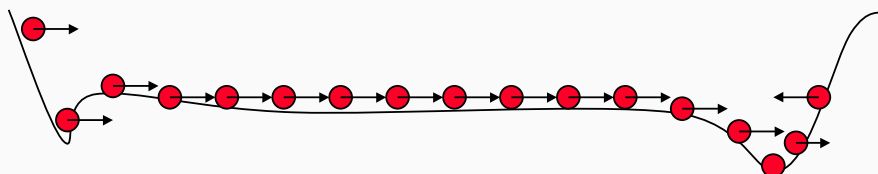
- StattdesquadratischenFehlers  $F = \sum_k (u_k - t_k)^2$  kann auch jedeandereFehlerfunktiongewähltwerden, bis auf den RekursionsanfangbleibtdieFehler-Rekursionwievorher.
- DasBP-Verfahrenkanndurch einenMomentum-Term  $\Delta w_{ik} = \alpha e_k x_i + \beta \Delta_{alt} w_{ik}$  nochstabilergestaltetwerden.
- WeightDecay:  $\Delta w_{ik} = \alpha e_k x_i - \gamma w_{ik}$  dabei wird das  $\gamma$  so eingerichtet, daß in der Summe keine Gewichtsvergrößerungeintritt oder daß große Gwichte „bestraft“ werden.
- DieLernkonstanten und die Output-Funktionen können für jedeSchicht oder für jedesNeuroneinzeln festgelegt werden.
- DieLernkonstanten können im Laufedes Trainings dynamischangepaßt werden.

## MomentumTerm

- DasBP-Verfahrenkanndurch einenMomentum-Term  $\Delta w_{ik} = \alpha e_k x_i + \beta \Delta_{alt} w_{ik}$  nochstabilergestaltetwerden.

- In dieser Trainingsvariante richtet sich die Gewichtsveränderung nicht nach dem Fehler, sondern nur nach dessen Vorzeichen.
- Die übliche Regel zur Gewichtsveränderung wird ersetzt durch  $\Delta w_{ik} = \alpha \cdot \text{sgn}(e_k) \cdot x_i$ .
- Dies bedeutet, daß die Fehlerfunktion nicht mehr der quadratische Fehler sondern nun noch der lineare Fehlerbetrag ist.
- Dieses Verfahren beseitigt die Probleme zu kleiner und zu großer Gradienten bei flachen Plateaus bzw. steilen Tälern. Allerdings kommt dann die richtige Wahl von  $\alpha$  eine tragende Bedeutung zu.

- Das BP-Verfahren kann durch einen Momentum-Term  $\Delta w_{ik} = \alpha e_k x_i + \beta \Delta_{\text{alt}} w_{ik} = \lambda((1-\beta)e_k x_i + \beta \Delta_{\text{alt}} w_{ik})$  noch stabiler gestaltet werden.
- Der Effekt dieses Verfahrens ist, daß bei der Gewichtsänderung starke Richtungsänderungen erschwert werden (Trägheit) und damit das Verfahren verstetigt wird.
- Mit dieser Methode können die Probleme flacher Plateaus und steiler Täler gemindert und sogar das Entkommen aus nicht zu steilen lokalen Minima ermöglicht werden.



# FlatSpotElimination

- Um zu vermeiden, daß der BP-Algorithmus aus flatspots (z.B. für sigmoid-Funktion  $\theta' = \theta(1-\theta)$  bei sehr hoher/niedriger Neuronenaktivität) nicht mehr entkommt, kann man  $\theta'$  durch  $\theta' + \varepsilon$ ,  $\varepsilon > 0$  ersetzen.
- Damit ist der Gradient auch an flatspots nicht ganz 0 und ein Trainingseffekt tritt auch an dieser Stelle ein.
- Für kleine Netze ist ein guter Effekt mit  $\varepsilon = 0.1$  erzielt worden, ob diese Methode aber auf sehr großen und vielschichtigen Netzen noch positive Wirkung zeigt ist noch nicht untersucht worden.

# WeightDecay

- Ähnlich wie extreme Neuronenaktivitäten sind auch extreme Gewichtswerte in neuronalen Netzwerken problematisch, weil sie tendenziell die gesamte Netzumgebung dominieren (Großmutter-Neuronen).
- Der Vorschlag, die Gewichtsänderung auf Kosten des Gesamtgewichts durchzuführen  

$$\Delta w_{ik} = \alpha e_k x_i - \gamma w_{ik}$$
führt zu einer Bestrafung zu hoher Gewichte, weil dies der abgewandelten Fehlerfunktion entspricht:  

$$E^{\text{neu}} = E + \gamma/2 \sum w^2$$
- $\gamma$  wird in der Regel konstant zwischen 0.005 und 0.03 gewählt, bisweilen aber auch (*aufwendig und nicht mehr lokal!*) so, daß die Summe aller Änderungen 0 ergibt:  $\gamma = \alpha \sum e_k x_i / \sum w_{ik}$ .

- Manche Mängel (kleine Änderungen in oberen Schichten) von BP können dadurch behoben werden, daß man den einzelnen Verbindungen eigene Schrittweiten  $\alpha_{ij}$  gibt.
  - $\alpha_{ij}$  jeweils konstant aber mit Distanz vom Output wachsend
  - $\alpha_{ij}$  schrumpft oder wächst jeweils um den Faktor  $\kappa$ ,  $\kappa^+$  ( $0 < \kappa < 1 < \kappa^+$ ) je nachdem, ob der Fehlergradient das Vorzeichen wechselt oder nicht. Typische Werte sind 0.5 und 1.05, insbesondere  $\kappa^- \kappa^+ < 1$ .
  - $\alpha_{ij}$  schrumpft oder wächst jeweils um den Faktor  $\kappa$ ,  $\kappa^+$  ( $0 < \kappa < 1 < \kappa^+$ ) je nachdem, ob der Fehlergradient absolut wächst bzw. das Vorzeichen wechselt oder nicht.
  - Delta-Bar-Delta-Regel  
 $\alpha_{ij}$  schrumpft oder wächst je nachdem, ob der Fehlergradient das Vorzeichen wechselt oder nicht, dabei schrumpft  $\alpha_{ij}$  exponentiell aber wächst nur linear.

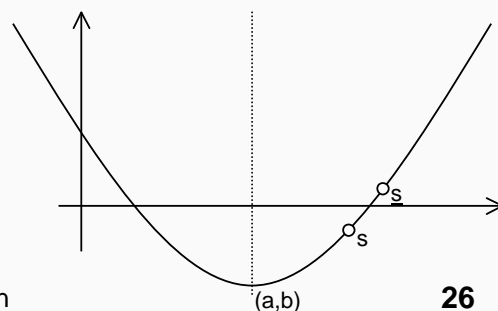
## höhere Verfahren

- Second-Order:  
Verwendung höherer Verfahren mit Hilfe der zweiten Ableitung (Jakobi-Matrix) führt zu beschleunigter Konvergenz aber höherer Anfälligkeit für lokale Minima.
- Quickprop:  
ist ein Beispiel für ein Verfahren 2. Ordnung.  
Gesucht: Tiefpunkt der an der Fehlerfläche angelegten Parabel mit Steigung  $S = e_k \theta'(a_k) x_i$  (Vorsicht bei  $S_{alt} = S!$ )  

$$\Delta w_{ik} = (S / (S_{alt} - S)) \cdot \Delta_{alt} w_{ik}$$
- $y = c(x-a)^2 + b$ ,  $y' = 2c(x-a)$   

$$\underline{s} - s = 2c(\underline{x} - x)$$
,  $\underline{x} - x = \Delta_{alt} w_{ik}$   

$$s = 2c(x-a)$$
,  $x-a = \Delta w_{ik}$



# Quickprop-Formeln

- $S = e_k \theta'(a_k) x_i$  //aktuelleSteigung  
 $\mu > 0$  //Schranke für die Gewichtsänderung
- $\Delta w_{ik} = G + P$  //Gradienten-undParabel-Term
- $G = \alpha e_k x_i + \beta w_{ik}$  falls  $\Delta_{alt} w_{ik} = 0$  oder  $\text{sgn}(S_{alt}) = \text{sgn}(S)$   
 $= 0$  sonst.
- $P = 0$  falls  $\Delta_{alt} w_{ik} = 0$   
 $= \left( \frac{S}{(S_{alt} - S)} \right) \cdot \Delta_{alt} w_{ik}$  falls  $|S / (S_{alt} - S)| \leq \mu$   
 $= \mu \cdot \Delta_{alt} w_{ik}$  sonst
- Der letzte Fall tritt insbesondere dann ein, wenn  $S_{alt} = S$  ist, also die Berechnung  $S / (S_{alt} - S)$  auf einen Divisionsfehler führt. Dies muß also vorher abgefangen werden.

# Verallgemeinerungen

- **selbstorganisiertes Backpropagation**
  - Statt jedem Input einen Target-Output beim Training zuzuordnen, wird nur eine bestimmte Form für Outputs erlaubt vorgegeben. Target ist dann die beste Näherung an tatsächlichen Output durch einander erlaubten Outputs.
- **Backpropagation mit Rückkopplungen.**
  - Netz mit Rückkopplungen, die aber nicht dem Training unterworfen werden
  - Backprop. through time **BPTT** durch Auffaltung des Netzes in mehrere Zeitschritte
  - recurrent Backpropagation **RBP** übliches Backpropagation jeweils im stabilen Zustand der Propagierung und der Backpropagierung

**selbstorganisiertes BP**

- Zum Training liegen nur Input-Daten ohne Soll-Output vor.
- Zufällige Vektoren werden zur Initialisierung der Verbindungsgewichte  $\mathbf{w}_k$  im MLP gewählt.
- Im Output-Raum wird eine Teilmenge (Code) von erlaubten Outputsgewählt
- Für jeden Input  $\mathbf{x}$  wird als Soll-Output der Code-Vektor  $\mathbf{c}$  gewählt, für den die Distanz zum Output minimal ist.
- Mit diesem (ggf. wechselnden) Soll-Output wird die Input-Schicht nach der BP-Regel trainiert.

**Selbstorganisiertes BP**

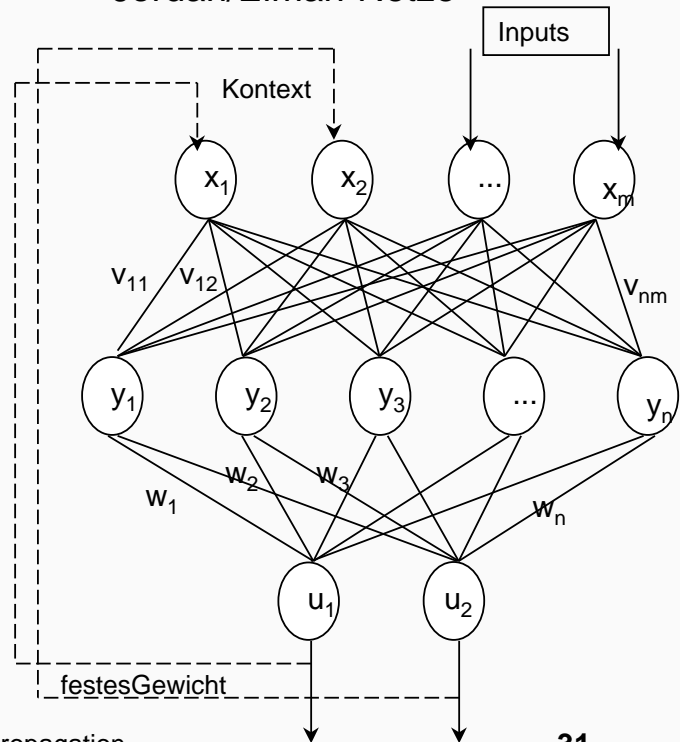
- In dieser Variante werden Outputs der redundanten Form  $(1, 0..0), (0, 1, 0..0) \dots (0..0, 1)$  angestrebt.
- Zu den Trainingsdaten gibt es keine Soll-Outputs, sondern für jeden Input wird als Soll-Output derjenige angenommen, der dem Ist-Output am nächsten kommt, dies läuft auf die Bestimmung des Maximums im Ist-Output hinaus.
- Zu diesem Output wird dann der übliche Fehler berechnet und das übliche Backpropagating in Gang gesetzt.
- Stattderspeziellen Form der Einheitsvektoren kann auch eine komplexere redundante Output-Form (fehlerkorrigierender Code) gewählt werden.

# Netze mit Zuständen

## Zeitrepräsentation in NN durch **Zustände**

- Zeitabhängigkeiten sind modellierbar durch weitere Neuronen (Zustände, Kontext) die den bisherigen Netzzustand speichern.
- Die Speicherung in Zuständen erfolgt durch nicht trainierbare Rückwärtsverbindungen
- kein Einschwingen in stabilen Zustand erforderlich

## Jordan/Elman-Netze



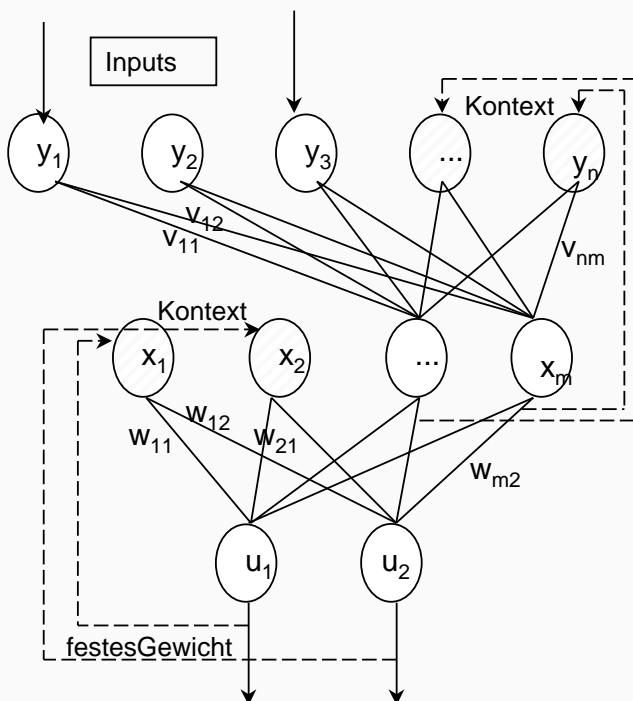
H.Werner/SS2000

Backpropagation

31

# Zustandsbestimmung

## • Kurzzeitgedächtnis

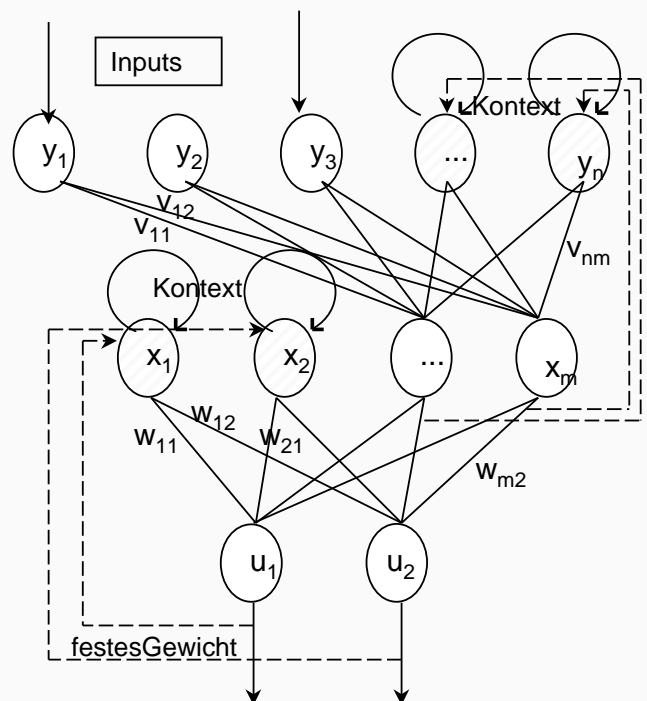


H.Werner/SS2000

Backpropagation

32

## • Langzeitgedächtnis

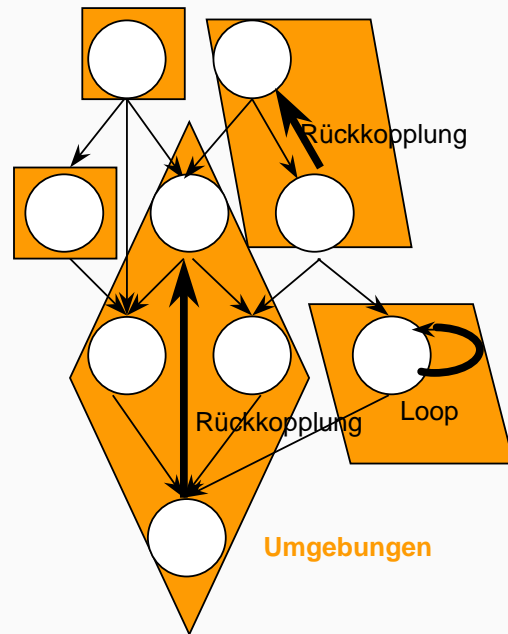




# Feed-Backward-Netze

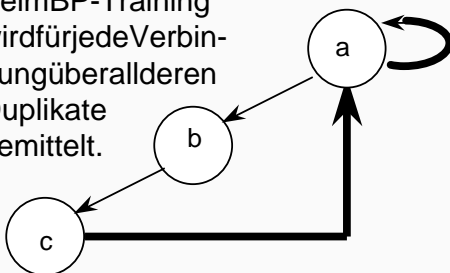
- In einem Netz mit Rückkopplungen versteht man unter einem **Zyklus** einen Weg, der zum Anfangsneuron zurückführt. Ein Spezialfall ist ein **Loop**, eine Verbindung eines Neurons mit sich selbst.
- Es gibt zwei Sorten von Verbindungen, die **zyklischen**, die in einem Zyklus vorkommen und die übrigen **vorwärts**-Verbindungen.
- Jedes Neuron hat eine **Umgebung**, das sind die Neuronen, die mit ihm in einem Zyklus liegen.
- Jedes Netzwerk kann so in Schichten eingeteilt werden, daß jede Umgebung ganz in einer Schicht liegt.

- Netz mit Rückkopplungen



# Entfaltung

- Ein Netz mit Rückkopplungen kann durch die Zeit entfaltet werden, indem man
  - Das Netz mehrfach untereinander wiederholt (**entfaltet**).
  - Jede Rückwärtsverbindung wird in das entsprechende Neuron der nächsten Kopie verlegt.
  - Beim BP-Training wird für jede Verbindung überall der entsprechende Duplikat gemittelt.



- entfaltetes Netzwerk

