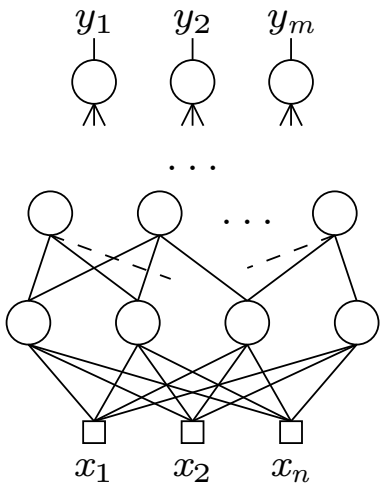
Mustererkennung: Neuronale Netze

D. Schlesinger – TUD/INF/KI/IS

Feed-Forward Netze



Output Schicht i_{max}

...

Zwischenschicht i

...

Zwischenschicht 1

Input Schicht 0

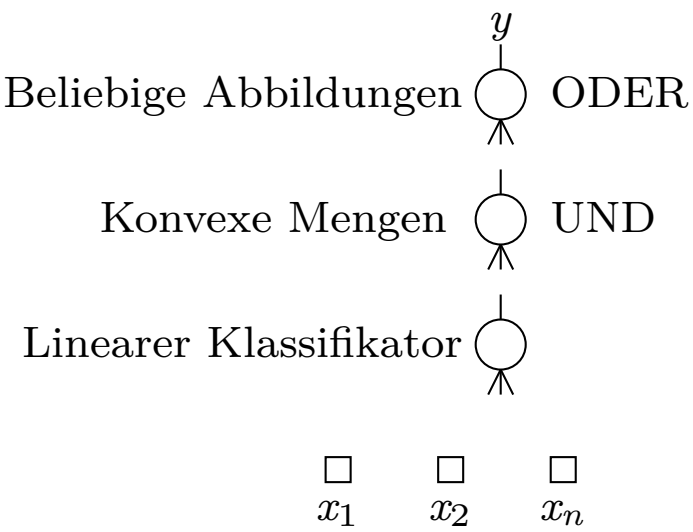
i – Nummer der Schicht
$i = 0$ – Input Schicht
$i = i_{max}$ – Output Schicht
j – Nummer des Neurons
y_i – Output der i-ten Schicht (Vektor)
y_{ij} – eine Komponente davon
$y_0 = x$ – Input Signal
w_i – Gewichtsmatrix, $i = 1 \dots i_{max}$
$w_{ijj'}$ – Gewicht, das (i, j) mit $(i-1, j')$ verbindet
b_{ij} – Schwellwerte

$$y_{ij} = f\left(\sum_{j'} w_{ijj'}\, y_{i-1\,j'} - b_{ij}\right)$$

Spezialfall: $m = 1$, Schwellwertneuronen – Realisiert eine Abbildung $\mathbb{R}^n \to \{0, 1\}$

Welche Abbildungen sind realisierbar?

Funktionalität



1 Schicht – ein einzelnes Neuron →
→ linearer Klassifikator

2 Schichten (was auf jeden Fall geht) –
– UND-Neuron in Output Schicht →
→ Schnitt der Halbräume → konvexe Gebiete
Wenn die Anzahl der Neuronen nicht beschränkt ist
– beliebige konvexe Gebiete (beliebig gut approximierbar)

3 Schichten – beliebige Abbildungen überhaupt als ODER über eine beliebige Anzahl der konvexen Gebiete

RBF-Neuron (Radial Basis Function) – „ersetzt eine Schicht“:

$$y = f(\|x - w\| - b)$$

Neuronales Netz mit nur einer Zwischenschicht aus RBF-Neuronen und einem ODER-Neuron als Output – beliebige Abbildungen $\mathbb{R} \to \{0, 1\}$!!!

Error Backpropagation

Das Verfahren zum Lernen Feed-Forward Netze mit Sigmoid-Neuronen – Gradient Methode
Gegeben: Lernstichprobe $\Big((x^1, k^1) \dots (x^l, k^l)\Big)$, $x^l \in \mathbb{R}^n$, $k^l \in \mathbb{R}$
Gesucht: Gewichte und Schwellwerte aller Neuronen

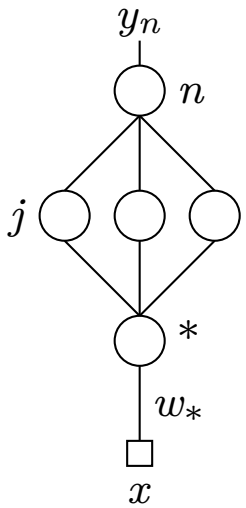
Sei $y(x; w, b)$ das Output des Netzes (mit aktuellen Parametern (w, b)) beim Input x
Die zu minimierende Zielfunktion ist:

$$F(w, b) = \sum_l \Big(k^l - y(x^l; w, b)\Big)^2$$

Ableitung (zunächst für nur ein Neuron und ein Paar (x, k))
– Anwendung der Kettenregel:

$$\frac{\partial F(w,b)}{\partial w_j} = \frac{\partial F}{\partial y} \cdot \frac{\partial y}{\partial y'} \cdot \frac{\partial y'}{\partial w_j} = (y - k) \cdot \frac{\exp(-y')}{\Big(1 + \exp(-y')\Big)^2} \cdot x_j = \delta \cdot d(y') \cdot x_j$$

Error Backpropagation



$$\frac{\partial F}{\partial w_*} =$$

$$\frac{\partial F}{\partial y_n} \cdot \frac{\partial y_n}{\partial y'_n} \cdot \left[\sum_j \frac{\partial y'_n}{\partial y_j} \cdot \frac{\partial y_j}{\partial y'_j} \cdot \frac{\partial y'_j}{\partial y_*} \right] \cdot \frac{\partial y_*}{\partial y'_*} \cdot \frac{\partial y'_*}{\partial w_*} =$$

$$\left[\sum_j \delta_n \cdot d(y'_n) \cdot w_{nj} \cdot d(y'_j) \cdot w_{j*} \right] \cdot d(y'_*) \cdot x =$$

$$\left[\sum_j \delta_j \cdot d(y'_j) \cdot w_{j*} \right] \cdot d(y'_*) \cdot x =$$

$$\delta_* \cdot d(y'_*) \cdot x$$

mit

$$\delta_j = \delta_n \cdot d(y'_n) \cdot w_{nj}$$

$$\delta_* = \left[\sum_j \delta_j \cdot d(y'_j) \cdot w_{j*} \right]$$

Error Backpropagation

Allgemein:
Berechne die „Fehler“ δ_{ij} in der i-ten Schicht aus den Fehlern $\delta_{i+1\ j}$ in der $i+1$-ten Schicht (bei den aktuellen Parametern des Netzes) – propagiere die Fehler zurück.

Algorithmus zur Berechnung des Gradienten für ein Paar (x, k):

1) Forward: Berechne alle y' und y (wende das Netz an),
   berechne den Output-Fehler $\delta_n = y_n - k$
2) Berechne die Fehler für Neuronen in Zwischenschichten:

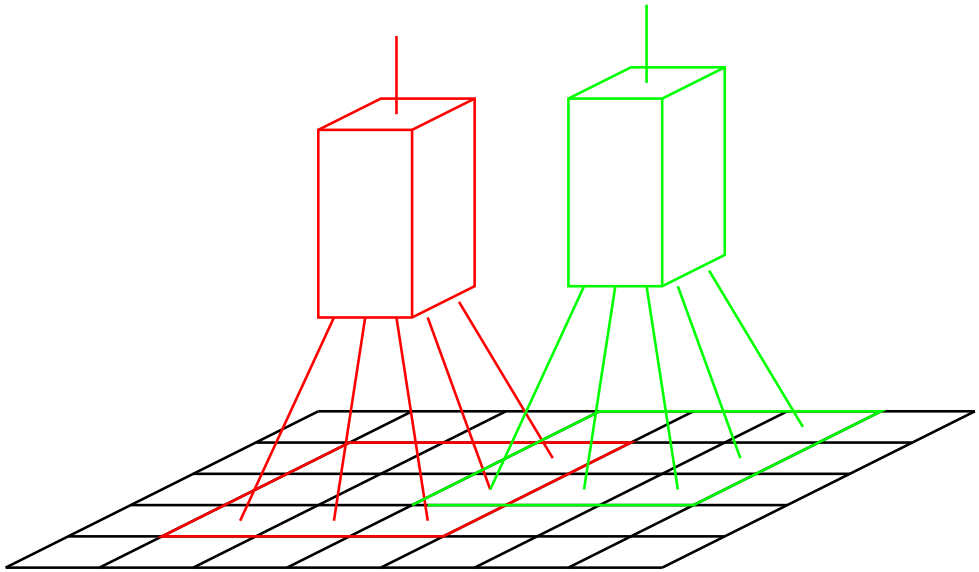
$$\delta_{ij} = \sum_{j'} \delta_{i+1\ j'} \cdot d(y_{i+1\ j'}) \cdot w_{i+1\ j'j}$$

3) Berechne den Gradient:

$$\frac{\partial F}{\partial w_{ijj'}} = \delta_{ij} \cdot d(y'_{ij}) \cdot y_{i-1\ j'}$$

Für mehrere Paare (x^l, k^l) – summiere die Gradienten.

Time Delay Neural Networks (TDNN)



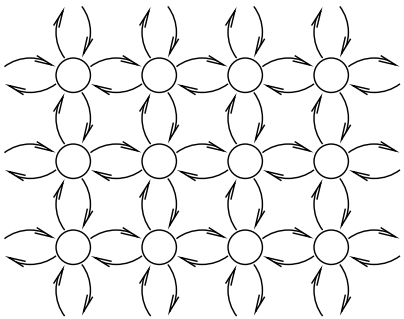
Feed-Forward Netz einer bestimmten Architektur:
Mehrere äquivalente „Teile“, die allerdings unterschiedliche **Rezeptive Felder** haben.

Die Output-Schicht eines Teiles in einer Position (i, j) des Bildes liefert Information über die Umgebung um (i, j) – Berechnung lokaler Merkmale.

Beim Lernen eines solches Netzes geht die Äquivalenz der Teile verloren (siehe Error Backpropagation). Um sie zu erhalten, wird eine zusätzliche Mittlung der Gewichte/Schwellwerte nach jedem Gradient-Schritt des Lernens notwendig.

Hopfield Netze

Es gibt eine symmetrische Nachbarschaftsrelation (Beispiel – Gitter).
Das Output jedes Neurons ist das Input für die benachbarten Neuronen.



$$y_r = f\Big(\sum_{r' \in N(r)} w_{rr'} \cdot y_{r'} + b_r \Big)$$

mit symmetrischen Gewichten, d.h. $w_{rr'} = w_{r'r}$.

Eine Konfiguration ist eine Zuordnung $y : D \to \{0, 1\}$ – „Zustandsfeld“.
Energie einer Konfiguration ist:

$$E(y) = \sum_{rr'} w_{rr'} \cdot y_r \cdot y_{r'} + \sum_{r} b_r \cdot y_r$$

Hopfield Netz mit externem Input x:

$$E(y, x) = \sum_{rr'} w_{rr'} \cdot y_r \cdot y_{r'} + \sum_{r} b_r \cdot y_r + \sum_{r} q(y_r, x)$$

Realisiert eine Abbildung $X \to Y$ nach dem Prinzip maximaler Energie:

$$y = \arg\max_{y'} E(y', x)$$

Dynamik des Netzes: ausgehend von einer Konfiguration $y^{(0)}$ werden Neuronen in andere Zustande entsprechend

$$y_r = f\Big(\sum_{r' \in N(r)} w_{rr'} \cdot y_{r'} + b_r \Big)$$

gesetzt. Somit steigt die Energie des Netzes

$$E(y) = \sum_{rr'} w_{rr'} \cdot y_r \cdot y_{r'} + \sum_r b_r \cdot y_r .$$

Man betrachte die neue Energie nach der Zustandsänderung eines einzelnes Neurons:

$$\begin{aligned} E^{(t+1)}(y) - E^{(t)}(y) = \\ = \Big(y_r^{(t+1)} - y_r^{(t)} \Big) \cdot \Big[\sum_{r' \in N(r)} w_{rr'} \cdot y_{r'} + \sum_r b_r \Big] \end{aligned}$$

Ist $[\cdot] > 0$, so wird y_r ins 1 gesetzt (laut oberste Formel)
$\Rightarrow$ die Energie wird nicht verringert.

Diese Dynamik ist die einfachste Methode zum finden der Zustandskonfiguration maximaler Energie (Iterated Conditional Modes).
Sie ist aber nicht global optimal !!!

Die Methode konvergiert zum lokalen Optimum der Energie. Selbst das nur dann, wenn die Zustände der Neuronen sequentiell geändert werden – bei der Änderung eines Neurons sind alle anderen fixiert (Sequenzielle Dynamik).

Im Falle paralleler Dynamik kann unter Umständen ein oszillierendes Verhalten entstehen. Beispiel: $b_1 = b_2 = 1$, $w_{12} = -2$

$$E(y_1, y_2) = y_1 - 2 \cdot y_1 \cdot y_2 + y_2$$

Die Suche nach dem Zustandsfeld optimaler Energie (bei gegebener Beobachtung x) ist eine schwierige (im Allgemeinen NP-vollständige) Aufgabe.
Energy Minimization Methods – bekannte polynomiell lösbare Klassen:

- Die Nachbarschaftsstruktur ist ein (k-breiter Baum). Spezialfall – Ketten. Dynamische Programmierung.
- Die Energie ist supermodular – entspricht positiven Gewichten $w_{rr'}$.

Es gibt viele approximative Lösungen für allgemeinen Fall.

Cohonen Netze

Selbstorganisierende Karte (Self Organizing Maps – SOM).

Eine Beispiel-Aufgabe:
Gegeben sei eine Menge der Datenpunkten in $\mathbb{R}^n$, die einem Objekt entsprechen (nach welchem schließlich gesucht wird). Zusätzlich sei bekannt, dass der Objekt bestimmte topologische Eigenschaften besitzt. Zum Beispiel ist der Objekt eine Untermannigfaltigkeit niedriger Dimension.

Beispiel 1: Der Objekt ist eine 1D-Linie im 2D, d.h. sie ist durch eine Menge der schwarzen Pixel im $\mathbb{R}^2$ dargestellt.
Beispiel 2: Gegeben sei die Menge der Punkte im 3D-Raum. Gesuch wird nach dem Mesh (ein Dreiecksnetz – eine 2D-Untermannigfaltigkeit).

Die Aufgabe ist, die gegebene Datenmente durch ein neuronales Netz vorgegebener Topologie „zu approximieren“.

Cohonen Netze

Cohonen Netze bestehen (meist) aus RBF-Neuronen so, dass jedes Neuron einer Untermenge des Input-Raums entspricht. Dies erfolgt durch geeignete Wahl der Parameter (z.B. des Zentruns des RBF-Neurons).

Die Menge der Neuronen ist mit einem Distanzmaß versehen, die der gewünschten Topologie entsprechen, d.h. für jedes Paar von Neuronen (r, r') gibt es einen Abstand $d(r, r')$.

Beispiel: die Neuronen sind die Knoten eines Graphen (z.B. eine Kette, wenn die gewünschte Topologie einem 1D-Objekt entspricht). Der Abstand $d(r, r')$ ist der kürzeste Weg von r nach r'.

Besonderheit: für die Neuronen gibt es keinen gewünschten Output – das unüberwachte Lernen.

Zusammenfassung:

- RBF-Neuronen, jedes für sein Teilraum verantwortlich – Fisher Klassifikator.
- Die Menge der Neuronen besitzt eine Topologie.
- Das unüberwachte Lernen.

Cohonen Netze

Lernalgorithmus (sequenzielle Variante):

1) Nehme zufällig ein Muster x aus der Lernstichprobe
2) Bestimme das „Gewinner-Neuron“:

$$r^* = \arg\min_r \|x - w_r\|$$

3) Bestimme die Umgebung des Gewinner-Neurons im Netz:

$$R = \{r | d(r, r') < \Theta\}$$

4) Aktualisiere die Gewichte aller Neuronen aus R:

$$w_r = w_r + (x - w_r) \cdot \eta\Big(t, d(r^*, r)\Big)$$

Varianten je nach Art der Funktion $\eta(t, d)$.
Generell ist η monoton fallend in t und d.

Ohne 3) und $d(r, r')$ – K-Means Algorithmus.

Parallele Variante:
– gehe über alle Datenpunkte, summiere Gradienten, wende sie anschließend an.

Cohonen Netze