5. Multilayer-Perzeptron

5.1 Nicht-lineare Probleme

Überblick:

- Motivation
- Vorverarbeitung zur Linearisierung des Problems
- Netze von mehreren Perzeptrons
- Zwei-Schicht-Netz für XOR

Motivation

Perzeptron: Lineare Trennbarkeit der Klassen

Adaline: Lineare Funktionsapproximation

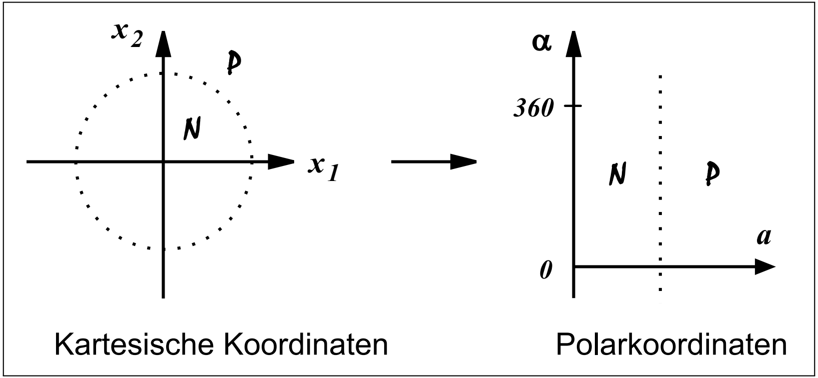
Gesucht: Lösungen für Nicht-lineare Trennbarkeit bzw. Nicht-lineare Funktionsapproximation !

Vorverarbeitung zur Linearisierung des Problems

1.Beispiel:

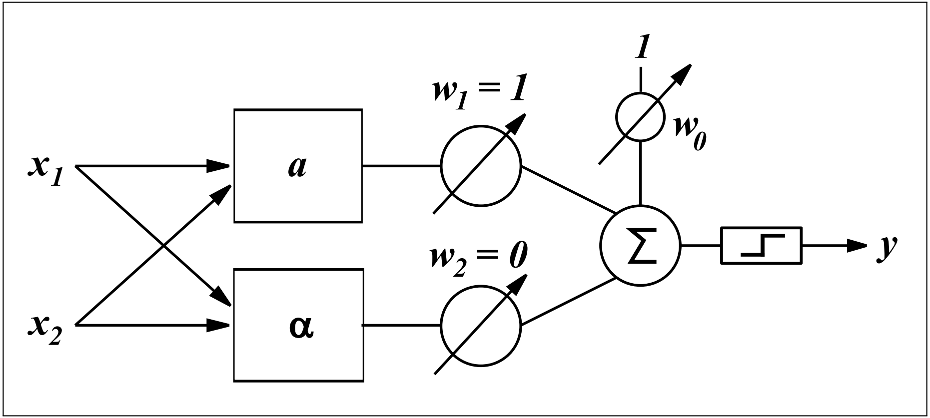
Nicht-lineare Transformation, Erhaltung der Dimension, Perzeptron

Transformation kartesische in Polarkoordinaten



Vorverarbeitung zur Linearisierung des Problems

Perzeptron entscheidet linear anhand Polarkoordinate $\boldsymbol{a}$

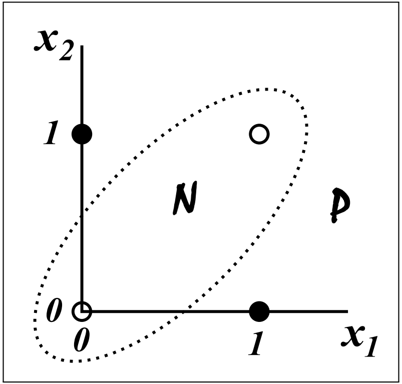


Vorverarbeitung zur Linearisierung des Problems

2.Beispiel:

Nicht-lineare Transformation, Erhöhung der Dimension, Perzeptron

Elliptische Trennfunktion im 2D-Originalraum bei f_{XOR}

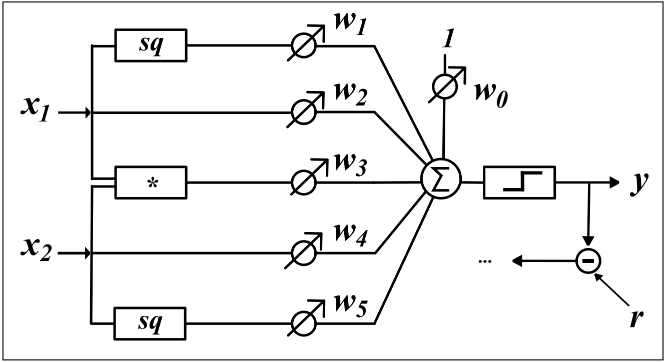


Vorverarbeitung zur Linearisierung des Problems

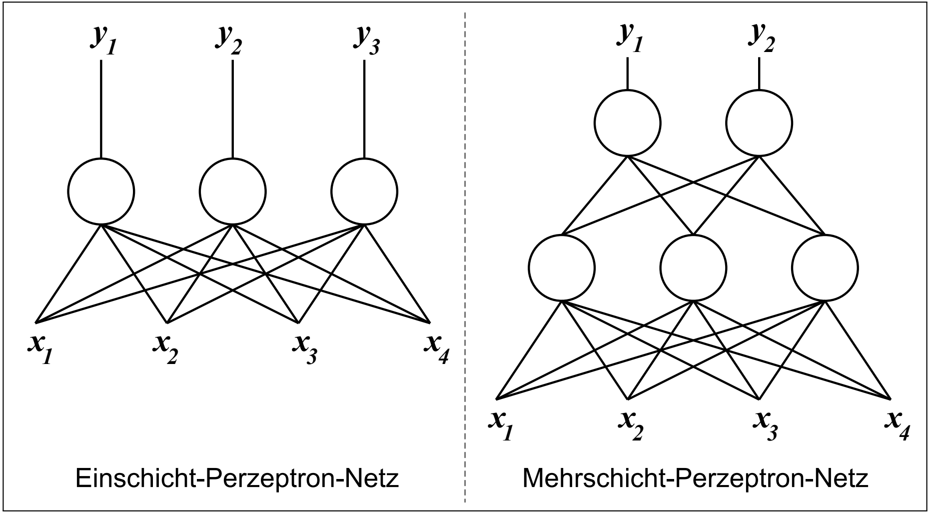
Allgemeine Gleichung für Ellipsen:

$$\boldsymbol{w_0 + w_1 x_1^2 + w_2 x_1 + w_3 x_1 x_2 + w_4 x_2 + w_5 x_2^2 = 0}$$

Lineare Trennung im 5D-Raum mit 4D-Hyperfläche

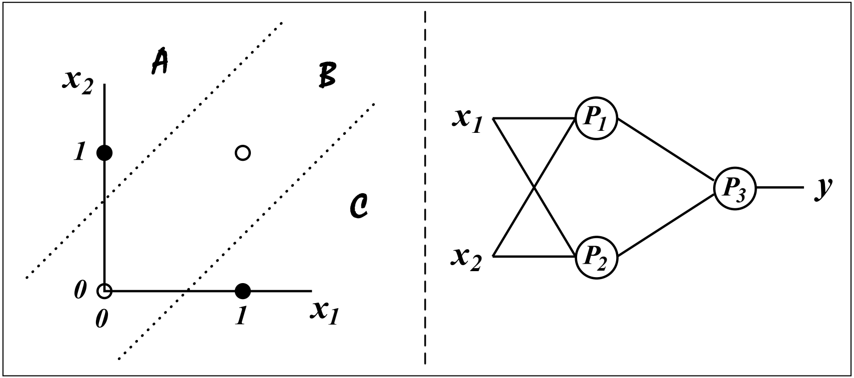


Netze von mehreren Perzeptrons



Zwei-Schicht-Netz für XOR

Bsp.: Durch Kombination von 3 linearen Trennfunktionen, realisiert durch 3 Perzeptrons in 2 Schichten, kann XOR realisiert werden.



Zwei-Schicht-Netz für XOR

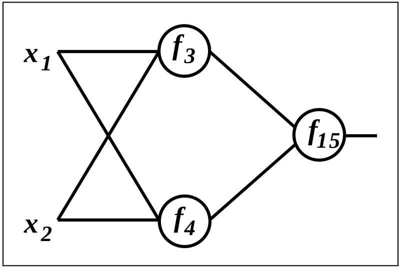
Alle Funktionen eines zwei-stelligen binären Inputs:

x_1	x_2	f_1	f_2	f_3	f_4	$\dots$	XOR	$\dots$	f_{15}	f_{16}
0	0	0	1	0	0		0		0	1
0	1	0	0	1	0		1		1	1
1	0	0	0	0	1		1		1	1
1	1	0	0	0	0		0		1	1

Zwei-Schicht-Netz für XOR

Drei binäre, lineare Funktionen, organisiert in 3 Schichten, realisieren XOR:

x_1	0	0	1	1
x_2	0	1	0	1
f_3	0	1	0	0
f_4	0	0	1	0
f_{15}	0	1	1	0



Zwei-Schicht-Netz für XOR

Kodierung von Teilbereichen des Eingaberaumes:

	$\boldsymbol{Y_{f_3}}$	$\boldsymbol{Y_{f_4}}$
$\boldsymbol{\mathcal{A}}$	1	0
$\boldsymbol{\mathcal{B}}$	0	0
$\boldsymbol{\mathcal{C}}$	0	1

Aus 4 Trainingselementen $\{(0,1),(0,0),(1,1),(1,0)\}$ entstehen durch erste Perzeptron-Schicht 3 Elemente $\{(1,0),(0,0),(0,1)\}$, die durch ein Perzeptron der zweiten Schicht beliebig trennbar sind. Nicht-Linearität durch Gesamtheit der Knoten der ersten Schicht erreicht, dies bewirkt Neukodierung.

5.2 Backpropagation-Lernen in MLPs

Rumelhart, 1986

Überblick:

- Symbole für Backpropagation-Lernen
- Herleitung Backpropagation-Formeln
- Zusammenfassung der Herleitung
- Algorithmus für Backpropagation-Lernen
- Beendigung des Algorithmus
- Resumee zum Backpropagation-Lernen

Symbole für Backpropagation-Lernen

$\boldsymbol{L}$	Zahl der Schichten
$\boldsymbol{N_\ell}$	Zahl der Knoten in Schicht $\boldsymbol{\ell}$
$\boldsymbol{M}$	Zahl der Trainingsvektoren
$\boldsymbol{x^m}$	$\boldsymbol{m}$-ter Trainingsvektor
$\boldsymbol{y_{\ell,j}}$	Output des $\boldsymbol{j}$-ten Knotens in Schicht $\boldsymbol{\ell}$, $\boldsymbol{0 \le \ell \le L}$, $\boldsymbol{0 \le j \le N_\ell}$
$\boldsymbol{y_{0,i}}$	$\boldsymbol{i}$-te Komponente des Input-Vektors ($\boldsymbol{y_{0,i} = x_i}$)

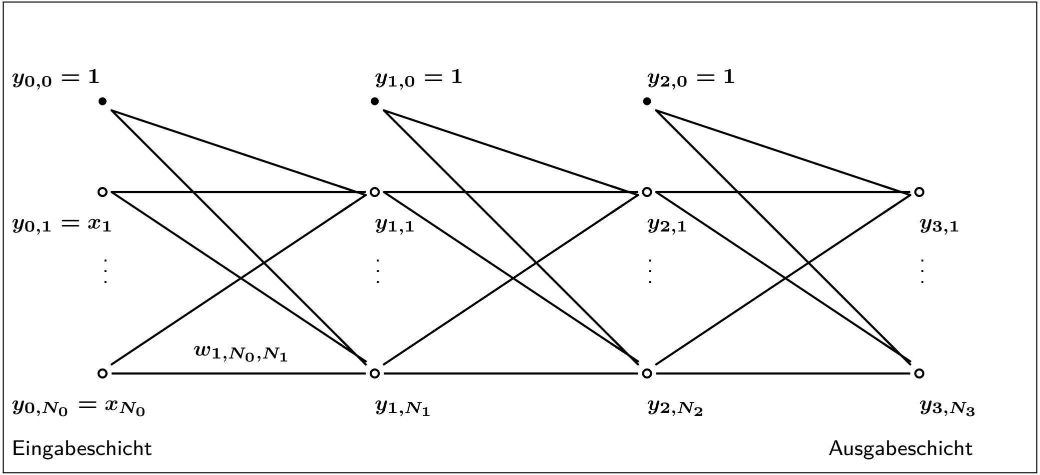
Symbole für Backpropagation-Lernen

$w_{\ell,i,j}$ Gewicht, welches den i-ten Knoten in Schicht $\ell - 1$ mit dem j-ten Knoten in Schicht ℓ verbindet.

$r_j(x^m)$ Geforderte Antwort des j-ten Output-Knotens für den m-ten Trainingsvektor,
d.h. bei Klassifikation, die gewünschte Wahrscheinlichkeit für Klasse c_j (üblicherweise 0 oder 1),
oder bei Regression, der gewünschte Funktionswert.

Symbole für Backpropagation-Lernen

Beispielhaft für ein MLP:



Herleitung Backpropagation-Formeln

Output des Knotens $\boldsymbol{j}$ in Schicht $\boldsymbol{\ell}$

$$\boldsymbol{y_{\ell,j} = f_\sigma(u_{\ell,j}) = f_\sigma\left(\sum_{i=0}^{N_{\ell-1}} w_{\ell,i,j} \cdot y_{\ell-1,i}\right)}$$

$$\boldsymbol{f_\sigma(u) = (1 + e^{-u})^{-1}}$$

Herleitung Backpropagation-Formeln

Fehlerfunktion des Gewichtsvektors $\boldsymbol{w}$,

$\boldsymbol{D(w) = \frac{1}{M} \sum_{m=1}^{M} d^m(w)}$,

mit $\quad \boldsymbol{d^m(w) = \frac{1}{2} \sum_{j=1}^{N_L} (y_{L,j}(x^m) - r_j(x^m))^2}$

ist Summe der quadratischen Fehler an $\boldsymbol{N_L}$ Output-Knoten.

Iterative Bestimmung der Gewichte durch Gradientenabstieg an der Fehlerfunktion:

$$\boldsymbol{w_{l,i,j}^{t+1} := w_{l,i,j}^{t} - \mu \cdot \left. \frac{\partial D(w)}{\partial w_{l,i,j}} \right|_{w_{l,i,j}^t}} \tag{1}$$

mit Lernrate $\boldsymbol{\mu}$.

Herleitung Backpropagation-Formeln

Im folgenden wird Zeitindex $\boldsymbol{t}$ bei $\boldsymbol{w}^t$ weggelassen. Ausdruck für partielle Ableitungen von $\boldsymbol{d}^m(\boldsymbol{w})$ bzgl. Gewichte:

$$\frac{\partial \boldsymbol{d}^m(\boldsymbol{w})}{\partial \boldsymbol{w}_{\ell,i,j}} = \frac{\partial \boldsymbol{d}^m(\boldsymbol{w})}{\partial \boldsymbol{y}_{\ell,j}} \cdot \frac{\partial \boldsymbol{y}_{\ell,j}}{\partial \boldsymbol{w}_{\ell,i,j}}$$

mit $\boldsymbol{y} = \boldsymbol{f}_{\boldsymbol{\sigma}}(\boldsymbol{u})$ folgt für

$$\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{w}} = \frac{\partial \boldsymbol{f}_{\boldsymbol{\sigma}}(\boldsymbol{u})}{\partial \boldsymbol{u}} \cdot \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{w}}$$

Herleitung Backpropagation-Formeln

somit

$$
\begin{aligned}
\frac{\partial y_{\ell,j}}{\partial w_{\ell,i,j}} &= \frac{\partial}{\partial w_{\ell,i,j}} \left[f_\sigma \left(\sum_{n=0}^{N_{\ell-1}} w_{\ell,n,j} \cdot y_{\ell-1,n} \right) \right] \\
&= f'_\sigma \left(\sum \ldots \right) \cdot \frac{\partial}{\partial w_{\ell,i,j}} \left[\sum \ldots \right] \\
&= f'_\sigma(u_{\ell,j}) \cdot y_{\ell-1,i} \\
&\overset{(\star)}{=} y_{\ell,j} \cdot (1 - y_{\ell,j}) \cdot y_{\ell-1,i} \;,
\end{aligned}
$$

mit $y_{\ell,j} = f_\sigma(u_{\ell,j})$

Herleitung Backpropagation-Formeln

Einschub an ($\star$):

$$f_\sigma'(u) = \frac{e^{-u}}{(1+e^{-u})^2}$$

$$f_\sigma(u) \cdot (1 - f_\sigma(u)) = \frac{1}{1+e^{-u}} \cdot \left(1 - \frac{1}{1+e^{-u}}\right) =$$

$$\frac{1}{1+e^{-u}} \cdot \frac{e^{-u}}{1+e^{-u}} = \frac{e^{-u}}{(1+e^{-u})^2}$$

Herleitung Backpropagation-Formeln

Einsetzen in ursprünglichen Ausdruck liefert:

$$\frac{\partial d^m(w)}{\partial w_{\ell,i,j}} = \frac{\partial d^m(w)}{\partial y_{\ell,j}} \cdot y_{\ell,i} \cdot (1 - y_{\ell,j}) \cdot y_{\ell-1,i} \tag{2}$$

Ausdruck $\frac{\partial d^m(w)}{\partial y_{\ell,j}}$ die *Sensitivität* von $d^m(w)$ bzgl. Output $y_{\ell,j}$.

Der Knoten (ℓ, j) reicht seinen Einfluß auf d^m durch alle Knoten der nachfolgenden Schichten weiter.

Herleitung Backpropagation-Formeln

$$\frac{\partial d^m(w)}{\partial y_{\ell,j}} = \sum_{n=1}^{N_{\ell+1}} \frac{\partial d^m(w)}{\partial y_{\ell+1,n}} \cdot \underbrace{\frac{\partial y_{\ell+1,n}}{\partial y_{\ell,j}}}_{\text{Kettenregel}} \tag{3}$$

$$= \sum_{n=1}^{N_{\ell+1}} \frac{\partial d^m(w)}{\partial y_{\ell+1,n}} \cdot \frac{\partial}{\partial y_{\ell,j}} \left[f_\sigma \left(\sum_{q=0}^{N_\ell} w_{\ell+1,q,n} \cdot y_{\ell,q} \right) \right]$$

Herleitung Backpropagation-Formeln

$$
\begin{aligned}
&= \sum_{n=1}^{N_{\ell+1}} \frac{\partial d^m(w)}{\partial y_{\ell+1,n}} \cdot f'_\sigma(u_{\ell+1,n}) \cdot \frac{\partial u_{\ell+1,n}}{\partial y_{\ell,j}} \\
&= \sum_{n=1}^{N_{\ell+1}} \frac{\partial d^m(w)}{\partial y_{\ell+1,n}} \cdot y_{\ell+1,n} \cdot (1 - y_{\ell+1,n}) \cdot w_{\ell+1,j,n}
\end{aligned}
\tag{4}
$$

Fortsetzung für $\frac{\partial d^m(w)}{y_{\ell+1,j}}$ bis Output-Schicht:

$$
\frac{\partial d^m(w)}{\partial y_{L,j}} = y_{L,j}(x^m) - r_j(x^m) \tag{5}
$$

Zusammenfassung der Herleitung

Variante Batch-Lernen:

Zuerst Summierung der Gradienten über alle Elemente der Trainingsmenge und dann Korrektur der Gewichte.

Hierbei Einsetzen der Gleichungen 2, 3, 4, 5 in 1.

Zusammenfassung der Herleitung

Variante Online-Lernen:

In diesem Fall erhält man die Trainingselemente nur einzeln. Gewicht $\boldsymbol{w_{\ell,i,j}^{t+1}}$ in Gleichung (1) wird dann nur auf Grundlage von einem Trainingsvektor aktualisiert, d.h.

$$\boldsymbol{w_{\ell,i,j}^{t+1} := w_{\ell,i,j}^{t} - \mu \cdot \left.\frac{\partial d^m(w)}{\partial w_{l,i,j}}\right|_{w_{l,i,j}^{t}}} \tag{6}$$

Algorithmus für Backpropagation-Lernen

a) Feed-forward-Berechnung:
   Propagierung des/der Inputs vorwärts.

b) Berechnung der Gradienten:
   Starten an Ausgabeschicht und Backpropagation zu den verborgenen Schichten.

c) Korrektur der Gewichte:
   Änderung der Gewichte, so daß negative Gradientenrichtung verfolgt wird.

Algorithmus für Backpropagation-Lernen

```
proc back_prop
{ initialise_weights
  repeat
  {
    choose_next_training_element
    assign_input_vector

    feed_forward
    compute_gradient
    update_weights
  }
  until (termination_condition_reached)
}
```

Algorithmus für Backpropagation-Lernen

```
proc assign_input_vector
{ for j = 0 to N[0]
    y[0,j]=x[j]
}

proc feed_forward
{ for l = 1 to L
    for j = 1 to N[l]
      y[l,j] = f_sigma(sum(
                        i=0,N[l-1],
                        w[l,i,j]*y[l-1,i]))
}
```

Algorithmus für Backpropagation-Lernen

```
proc compute_gradient
{ for l = L to 1
  { for j = 1 to N[l]
    { if l=L then
        /* siehe Gleichung (5) */
        e[L,j] = y[L,j] - r[j]
      else
        /* siehe Gleichung (4) */
        e[l,j] = sum(
                   n=1,N[l+1],
                   e[l+1,n]*y[l+1,n] *
                   (1-y[l+1,n])*w[l+1,j,n])
```

Algorithmus für Backpropagation-Lernen

```
        for i = 1 to N[l-1]
          /* siehe Gleichung (2) */
          g[l,i,j] = e[l,j]*y[l,j]*
                     (1-y[l,j])*y[l-1,i]
      }
    }
}
```

Algorithmus für Backpropagation-Lernen

```
proc update_weights
{ for l = 1 to L
    for j = 1 to N[l]
      for i = 1 to N[l-1]
        /* siehe Gleichung (6) */
        w[l,i,j](t+1) = w[l,i,j](t) -
                              mu * g[l,i,j]
}
```

Erläuterung zu Symbolen:

$\texttt{e[l,j]} = \frac{\partial d^m(w)}{\partial y_{\ell,j}}$ und $\texttt{g[l,i,j]} = \frac{\partial d^m(w)}{\partial w_{\ell,i,j}}$

Beendigung des Algorithmus

- Falls kleiner Gradientenbetrag der Fehlerfunktion, d.h. $\left|\frac{\partial \boldsymbol{d}^m(\boldsymbol{w})}{\partial \boldsymbol{w}_{\ell,i,j}}\right| <$ Schwellenwert,
- oder falls kleiner Betrag des Fehlers, d.h. $|\boldsymbol{d}^m(\boldsymbol{w})| <$ Schwellenwert,
- oder nach fester Anzahl von Iterationen.

Hinweis: Gradientenabstieg führt nur zu lokalem Minimum der Fehlerfunktion.

Resumee zum Backpropagation-Lernen

- MLP-Lernen (Backpropagation-Lernen) wird realisiert durch Gradientenabstieg an der Fehlerfunktion.
- Fehlerfunktion ist wegen Verwendung der Sigmoid-Funktion nicht mehr quadratisch in den Gewichten, sondern komplexer.

$$\boldsymbol{D(w) := \frac{1}{2} \sum_{m=1}^{M} (r^m - f_\sigma(w^T x^m))^2}$$

- Gradientenabstieg findet eventuell nur lokales Minimum, statt globales.
- Gradientenabstieg ist nicht sehr effizient, besser wäre z.B. der sog. konjugierte Gradientenabstieg.

5.3 Funktionale Fähigkeiten von MLPs

Überblick:

- Nicht-lineare Klassifikation
- Sigmoid-Funktion zur Klassifikation
- Stetige, nicht-lineare Regression
- Identität als Aktivierungsfunktionen
- MLP realisiert Logik-Funktionen
- MLP realisiert Datenkompression

Nicht-lineare Klassifikation

1-Schicht-MLP kann lineare Entscheidungsgrenzen lernen (Halbräume).

2-Schicht-MLP kann stückweise lineare Entscheidungsgrenzen lernen (konvexe Räume).

3-Schicht-MLP lernt nicht-konvexe Räume.

Dabei ist:

Input-Vektor $\in \mathbb{R}^{\boldsymbol{I}}$, Output-Vektor $\in [\mathbf{0}, \mathbf{1}]^{\boldsymbol{K}}$,

Reelle Zahlen $\mathbb{R}$, Intervall $[\mathbf{0}, \mathbf{1}] \subset \mathbb{R}$,

Dimension $\boldsymbol{I}$ des Eingaberaums, Dimension des Ausgaberaums ist Anzahl $\boldsymbol{K}$ der Klassen.

Sigmoid-Funktion zur Klassifikation

Annahme: MLP für eine Klassifikationsaufgabe (2-Klassen-Problem)

Frage: Unter welcher Bedingung realisiert ein MLP einen Klassifikator, dessen Klassifikationsergebnis der a posteriori Wahrscheinlichkeit für eine Klasse entspricht ?

Formal muss also das Ergebnis der Sigmoid-Funktion identisch sein mit dem Ergebnis der Bayes-Formel:

$$\begin{aligned} P(c^1|x) &= \frac{P(x|c^1)P(c^1)}{P(x|c^1)P(c^1) + P(x|c^2)P(c^2)} \\ &\stackrel{!}{=} \frac{1}{1+e^{-u}} \end{aligned}$$

Sigmoid-Funktion zur Klassifikation

Man setze nun die Definition $u := \ln \frac{P(x|c^1)P(c^1)}{P(x|c^2)P(c^2)}$ in die Sigmoid-Funktion f_σ ein.

$$\frac{1}{1+e^{-u}} = \frac{1}{1+e^{-\left(\ln \frac{P(x|c^1)P(c^1)}{P(x|c^2)P(c^2)}\right)}} = \frac{1}{1+e^{\ln \frac{P(x|c^2)P(c^2)}{P(x|c^1)P(c^1)}}}$$

$$= \frac{1}{1+\frac{P(x|c^2)P(c^2)}{P(x|c^1)P(c^1)}} = \frac{1}{\frac{P(x|c^1)P(c^1)+P(x|c^2)P(c^2)}{P(x|c^1)P(c^1)}}$$

$$= \frac{P(x|c^1)P(c^1)}{P(x|c^1)P(c^1)+P(x|c^2)P(c^2)}$$

Sigmoid-Funktion zur Klassifikation

Es resultiert also damit gerade die Bayes-Formel, und somit die a posteriori Wahrscheinlichkeit für eine Klasse.

Falls die Gewichte so adaptiert wurden, dass bei Vorwärtspropagierung eines Input-Vektors der resultierende Wert von $\boldsymbol{u}$ identisch zum Ergebnis der obigen Formel ist, dann kann bei Verwendung der Sigmoid-Funktion $\boldsymbol{f_\sigma}$ als Aktivierungsfunktion der Output des MLP als a posteriori Wahrscheinlichkeit interpretiert werden.

Das MLP-Lernen führt i.A. nicht zu diesem gewünschten Wert von $\boldsymbol{u}$.

Hinweis auf Kapitel 6: Das Lernen eines RBF-Netzes wird dagegen gerade so realisiert, dass a posteriori Wahrscheinlichkeiten resultieren.

Stetige, nicht-lineare Regression

Durch Verwendung einer stetigen, nicht-linearen Aktivierungsfunktion, hier Sigmoid-Funktion $\boldsymbol{f_\sigma}$, können stetige, nicht-lineare Regressionsfunktionen approximiert werden.

Dabei ist: Input-Vektor $\in \mathbb{R}^{\boldsymbol{I}}$, Output-Vektor $\in \mathbb{R}^{\boldsymbol{O}}$, Reelle Zahlen $\mathbb{R}$, Dimension $\boldsymbol{I}$ des Eingaberaums, Dimension $\boldsymbol{O}$ der Ausgabe.

Satz zur Universellen Approximation (Cybenko, 1989): Ein 2-Schicht-Multi-Layer-Perzeptron erzeugt beliebig gute Approximationen für stetige, nicht-lineare Funktionen.

Identität als Aktivierungsfunktionen

Annahme: Mehrschichtiges Netz und Identität als Aktivierungsfunktion.

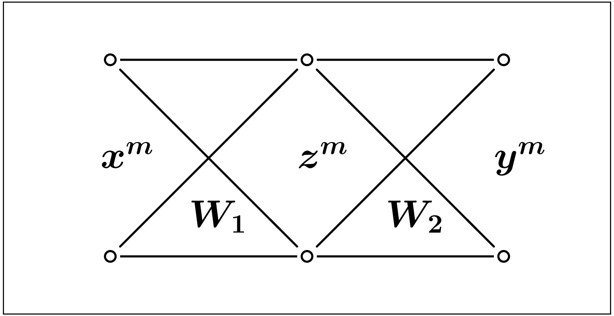
Gegeben: z.B. ein zweischichtiges Netz

Eingabeschicht: $\boldsymbol{x}^m = (\boldsymbol{x}_1^m, \ldots, \boldsymbol{x}_{N_0}^m)$

Verborgene Schicht: $\boldsymbol{z}^m = (\boldsymbol{z}_1^m, \ldots, \boldsymbol{z}_{N_1}^m)$

Ausgabeschicht: $\boldsymbol{y}^m = (\boldsymbol{y}_1^m, \ldots, \boldsymbol{y}_{N_2}^m)$

Identität als Aktivierungsfunktionen



$$z^m = x^m \cdot W_1; \quad y^m = z^m \cdot W_2 \quad = x^m \cdot \underbrace{W_1 W_2}_{=:W} = x^m \cdot W$$

Resumee: Falls als Aktivierungsfunktion die Identität gewählt wird, dann sind verborgene Schichten obsolet.

Identität als Aktivierungsfunktionen

Falls sämtliche Berechnungselemente von allen Schichten lediglich lineare Assoziatoren sind, dann kann insgesamt nur eine lineare Regression erreicht werden, d.h. ein einschichtiges Netz ist ausreichend.

Dagegen wird in Mehrschichtnetzen typischerweise eine einfache Nicht-Linearität der Berechnungselemente verwendet, um insgesamt eine komplexe nicht-lineare Klassifikation oder Regression zu erhalten.

MLP realisiert Logik-Funktionen

Jede Logik-Funktion hat eine äquivalente, disjunktive oder konjunktive Normalform.

Somit reicht eine verdeckte Schicht, um eine beliebige Logik-Funktion zu implementieren.

Dabei ist: Input-Vektor $\in \mathbb{B}^{\boldsymbol{I}}$, Output-Wert $\in \mathbb{B}$, Boole'sche Menge $\mathbb{B}$, Dimension $\boldsymbol{I}$ des Eingaberaums.

MLP realisiert Datenkompression

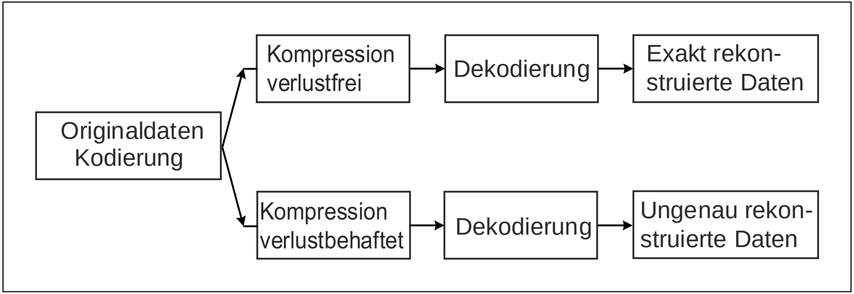
Problem: Suche einer effektiven Repräsentation (Kodierung) von Daten. Notwendig ist ein geeignetes Basissystem, in welchem eine Reduktion von Redundanzen in den Daten erreicht wird.

Kompression: Wenn man in einem Basissystem durch Reduktion von Redundanzen eine sparsamere Repräsentation der Daten hat als im kanonischen Basissystem.

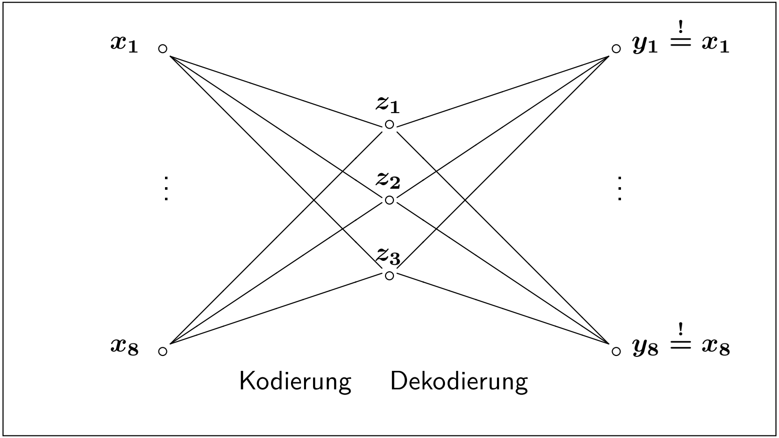
MLP realisiert Datenkompression

Kompressionsarten: Verlustfreie oder verlustbehaftete Transformation der Standardrepräsentation der Daten.

Verlustfreie Kompression heisst auch Autocodierung.



MLP realisiert Datenkompression



Training: Netz soll Eingabe unverändert weitergeben.

Anwendung: Weglassen des Dekodierungsabschnitts.