Softcomputing

Einsatz zum Lösen von Problemen,

- die unvollständig beschrieben sind
- die keine eindeutige Lösung haben
- für die keine effizienten exakten Algorithmen bekannt sind

einige Ansätze:

- Fuzzy-Logik, probabilistische Logik
- Maschinelles Lernen
- Künstliche neuronale Netze
- Evolutionäre Algorithmen
- Schwarm-Intelligenz

(Natürliches und) Maschinelles Lernen

(Schrittweise) Änderung eines Systems (Verfahrens zur Problemlösung), so dass es bei der zukünftigen Anwendung dasselbe oder ähnliche Probleme besser löst.

- Aufgaben (Problem): Menge von Eingaben
- Aufgabeninstanz: Eingabe
- Lösung der Instanz: Ausgabe
- Bewertung der Lösung: Zuordnung Lösung $\rightarrow$ Güte

Schritte bei der Lösung von Aufgabeninstanzen mit Lerneffekt:
Schüler (System) führt wiederholt aus:

1. verwendet ein Lösungsverfahren V für diese Aufgabe
2. bestimmt eine Lösung l der gegebenen Aufgabeninstanz
3. bestimmt (oder erfährt) eine Bewertung dieser Lösung l
4. modifiziert das Lösungsverfahren V zu V', um (in Zukunft) Lösungen mit besseren Bewertungen zu finden
5. wendet im nächsten Schritt zur Lösung dieser Aufgabe das Lösungsverfahren V' an

Lernen: Schritte 3 und 4

Lernverfahren

Lernen durch

- Auswendiglernen (gegebener Beispiele)
- Nachahmen
- Anleitung (Anweisungen)
- logische Ableitung neuer Lösungsverfahren
- Analogie (zu gegebenen Beispielen)
  anhand Ähnlichkeit
- Erfahrung (durch gegebene Beispiele)
  Fähigkeit zur Verallgemeinerung
- Probieren und Beobachten
  (Erzeugen eigener Beispiele)

nach Art des Lernenden:

- natürliches Lernen
- maschinelles (künstliches) Lernen

Lernen durch gegebene Beispiele

nach der zum Lernen verwendbaren Information:

überwachtes Lernen (supervised learning)

- korrigierendes Lernen (corrective learning)
- bestärkendes Lernen (reinforcement learning)

unüberwachtes Lernen (unsupervised learning)

gewünschte Eigenschaften des Löseverfahrens:

- Korrektheit
  der Lösungen für die gegebenen Beispiele
- Generalisierung
  „sinnvolle“ Lösungen für ähnliche Aufgaben

Korrigierendes Lernen

Trainingsmenge: Menge von Paaren (Eingabe, Ausgabe)
(partielle Funktion an Stützstellen)

Lernziel: (möglichst einfache) Funktion, die an den Stützstellen mit der Trainingsmenge übereinstimmt

Rückmeldung: Trainer sagt nach jedem Lernschritt die korrekte Ausgabe.

Prinzip: Lernen durch Nachahmen (mit Korrektur)

Anwendung z.B. bei

- Klassizierung (Zuordnung von Objekten / Fällen zu Klassen, abhängig von den Merkmalen der Objekte)
  z.B. Zuordnung Sensorwerte $\rightarrow$ Alarmklasse
  Trainingsmenge ist
  Menge von Paaren (Objekteigenschaften, Klasse)
- Lernen von Funktionen: Trainingsmenge ist
  Menge von Paaren (Parameter, Funktionswert)

Bestärkendes Lernen (reinforcement learning)

Trainingsmenge: Menge von Paaren (Eingabe, Erfolg $\in \{$ja, nein$\}$)

Lernziel: (möglichst einfache) Funktion, die den Stützstellen korrekte Werte zuordnet

Rückmeldung: Trainer sagt nach jedem Lernschritt, ob die Ausgabe korrekt war.

Idee: Lernen durch Probieren

- Klassifizierung: Trainingsmenge ist
  Menge von Objekten (mit ihren Eigenschaften)
  Bewertung der Lösung: ja, falls Zuordnung zur korrekten Klasse, sonst nein
- Lernen von Plänen (Anlagestrategien, Bewegungsabläufe usw.)
  z.B. Steuern eines autonomen Fahrzeuges
  Trainingsmenge: Strecke(n),
  Folge von Paaren (Sensordaten, Steuersignale)
  Bewertung der Lösung: ja, falls Plan zum Erfolg geführt hat (z.B. Fahrzeug fährt $> n$ km ohne Eingriff) , sonst nein

Unüberwachtes Lernen

Trainingsmenge: Menge von Eingaben

Lernziel:
- Gruppierung ähnliche Muster
- oft auch topologisch sinnvolle Anordnung

Idee: Lernen ohne Trainer (ohne Rückmeldung)

- Entdecken von Strukturen
- Selbstorganisation von Objekten zu Gruppen
  (mit gemeinsamen Merkmalen, typische Vertreter)
- topologieerhaltende Abbildungen
  (z.B. Körperteile $\rightarrow$ Gehirnregionen)
- Assoziation (z.B. in Schrifterkennung)

Neuronale Netze

Neuron – Nerv (griechisch)

Modellierung und Simulation der Strukturen und Mechanismen im Nervensystem von Lebewesen

Biologisches Vorbild	Mathematisches Modell
Nervenzellen (Neuronen)	künstliche Neuronen
Struktur (eines Teiles) eines Nervensystems	künstliche neuronale Netze (KNN) unterschiedlicher Struktur
Aktivierung von Neuronen, Reizübertragung	künstlichen Neuronen zugeordnete Funktionen
Anpassung (Lernen)	Änderungen verschiedener Parameter des KNN

Natürliche Neuronen

ZNS besteht aus miteinander verbundenen Nervenzellen (Neuronen)

Struktur eines Neurons:

- Zellkörper
- Dendriten
- Synapsen (verstärkende, hemmende)
- Axon

Natürliche Neuronen – Funktionsweise

Informationsübertragung durch elektrochemische Vorgänge:

- aktivierte Zelle setzt an Synapsen Neurotransmitter frei,
- Neurotransmitter ändern die Durchlässigkeit der Zellmembran für Ionen an den Dendriten der empfangenden Zelle,
- Potential innerhalb der empfangenden Zelle ändert sich durch diffundierende Ionen,
- überschreitet die Summe der an allen Synapsen entstandenen Potentiale (Gesamtpotential) der Zelle einen Schwellwert, entsteht ein Aktionsspotential (Zelle feuert),
- Aktionspotential (Spannungsspitze) durchquert das Axon (Nervenfaser) zu den Synapsen zu Nachbarzellen,
- aktivierte Zelle setzt an Synapsen Neurotransmitter frei, usw.

Stärke der Information durch Häufigkeit der Spannungsspitzen (Frequenzmodulation).

Eigenschaften natürlicher neuronaler Netze

- geringe Taktrate $10^{-}3$ s
- parallele Arbeit sehr vieler (10^{11}) Neuronen
- Neuronen sehr stark miteinander vernetzt
  (ca. 10 000 Nachbarn)
- Verarbeitungseinheit = Speicher

Vorteile:

- hohe Arbeitsgeschwindigkeit durch Parallelität,
- Funktionsfähigkeit auch nach Ausfall von Teilen des Netzes,
- Lernfähigkeit,
- Möglichkeit zur Generalisierung

Ziel: Nutzung dieser Vorteile zum Problemlösen
durch Wissensrepräsentation als künstliche neuronale Netze

Natürliche Neuronen – Lernen

Speicherung von Informationen durch Anpassung der Durchlässigkeit (Leitfähigkeit) der Synapsen

- Regel von Hebb (1949):
  Synapsen zwischen gleichzeitig aktiven Zellen werden immer durchlässiger (Reizschwelle wird verringert),
  Verbindung an dieser Synapse wird stärker
- lange nicht benutzte Synapsen verlieren mit der Zeit ihre Durchlässigkeit
  Verbindung an dieser Synapse wird schwächer.

Anwendungen künstlicher neuronaler Netze

Anwendungsgebiete:

- Bildverarbeitung, z.B.
  - Objekterkennung
  - Szenenerkennung
  - Schrifterkennung
  - Kantenerkennung
- Medizin, z.B. Auswertung von Bildern, Langzeit-EKGs
- automatische Spracherkennung
- Sicherheit, z.B. Biometrische Identifizierung
- Wirtschaft, z.B. Aktienprognosen, Kreditrisikoabschätzung
- Robotik, z.B. Lernen vom Bewegungsabläufen
- Steuerung autonomer Fahrzeuge

Geschichte künstlicher neuronaler Netze

- 1943, Warren McCulloch, Walter Pitts:
  A logical calculus of the ideas immanent in nervous activity
- 1949, Donald O. Hebb: Lernmodell
  The organization of behaviour
- 1957 Frank Rosenblatt: Perzeptron (1 Schicht)
  erster Neurocomputer MARK 1
  (Ziffernerkennung in 20×20-Bildsensor)
- 1969, Marvin Minsky, Seymour Papert: Perceptrons
- 1971 Perzeptron mit 8 Schichten
- 1974 Backpropagation (Erfindung)
- 1982, Teuvo Kohonen: selbstorganisierene Karten
- 1982, John Hopfield: Hopfield-Netze
- 1985, Backpropagation (Anwendung)
- 1997, long short-term memory (Erfindung)
- 2000, Begriff Deep Learning für KNN, Faltungsnetze (CNN)
- 2006, long short-term memory (Anwendung)
- 2009, verstärkt Training mit GPUs
- 2017, AlphaZero, ...

Künstliche Neuronen: McCulloch-Pitts-Neuron ohne Hemmung

einfaches abstraktes Neuronenmodell von
McCulloch und Pitts, 1943

Aufbau eines künstlichen Neurons u (Tafel)

Eingabe:	$x = (x_1, \ldots, x_{m_u}) \in \{0,1\}^{m_u}$	(ankommende Reize)
Schwellwert:	$\theta_u \in \mathbb{R}$	(Reizschwelle)
Ausgabe:	$f(x_1, \ldots, x_{m_u}) \in \{0,1\}$	(weitergegebener Reiz)

Parameter eines McCulloch-Pitts-Neurons u ohne Hemmung:

- m_u: Anzahl der (erregenden) Eingänge
- θ_u: Schwellwert

McCulloch-Pitts-Neuron ohne Hemmung: Funktionen

Eingangsfunktion des Neurons u: I_u: $\{0,1\}^{m_u} \to \mathbb{R}$ mit

$$I_u(x_1, \ldots, x_{m_u}) = \sum_{i=1}^{m_u} x_i$$

(Summe aller erregenden Eingänge des Neurons u)
Aktivierungsfunktion des Neurons u (abhängig vom Schwellwert θ_u): A_u: $\mathbb{R} \times \mathbb{R} \to \{0,1\}$ mit

$$A_u(\theta_u, v) = \begin{cases} 1 & \text{falls } v \geq \theta_u \\ 0 & \text{sonst} \end{cases}$$

(Stufenfunktion mit Stufe bei θ_u)
Ausgabefunktion des Neurons u: O_u: $\{0,1\} \to \{0,1\}$ mit

$$O_u(v) = v$$

(Identität)

McCulloch-Pitts-Neuron ohne Hemmung: Berechnung

vom Neuron u berechnete Funktion: $f_u: \{0,1\}^{m_u} \to \{0,1\}$ mit

$$\begin{aligned} f_u(x_1, \ldots, x_{m_u}) &= O_u(A_u(\theta_u, I_u(x_1, \ldots, x_{m_u}))) \\ &= \begin{cases} 1 & \text{falls } \sum_{i=1}^{m_u} x_i \geq \theta_u \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

m_u-stellige Boolesche Funktion

McCulloch-Pitts-Neuron ohne Hemmung: Beispiele

elementare Boolesche Funktionen $\vee, \wedge$

mehrstellige $\vee, \wedge$

Existiert zu jeder Booleschen Funktion $f : \{0,1\}^n \to \{0,1\}$ ein McCulloch-Pitts-Neuron ohne Hemmung, welches f berechnet?

Nein, nur monotone Boolesche Funktionen,
z.B. $\neg$ nicht

Warum?

Geometrische Interpretation

Jedes McCulloch-Pitts-Neuron u mit m_u Eingängen teilt die Menge $\{0,1\}^{m_u}$ in zwei Teilmengen:

$$\begin{aligned} f_u^{-1}(1) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 1\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \sum_{i=1}^{m_u} \geq \theta_u\} \end{aligned}$$

und

$$\begin{aligned} f_u^{-1}(0) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 1\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \sum_{i=1}^{m_u} < \theta_u\} \end{aligned}$$

geometrische Interpretation als Teilräume des R^m
Grenze zwischen beiden Bereichen:
$(m_u - 1)$-dimensionaler Teilraum $\sum_{i=1}^{m_u} x_i = \theta$
parallele Schnitte (abhängig von θ)

Geometrische Interpretation: Beispiele

Beispiele:

- Neuron u mit $m_u = 2$ Eingängen und Schwellwert $\theta_u = 1$

$$f_u(x_1, x_2) = \begin{cases} 1 & \text{falls } x_1 + x_2 \geq 1 \\ 0 & \text{sonst} \end{cases}$$

Bereich der x_1, x_2-Ebene mit $f_u(x_1, x_2) = 1$ ist die Halbebene mit $x_2 \geq 1 - x_1$.

$x_2 = g(x_1) = 1 - x_1$ ist eine lineare Trennfunktion zwischen den Halbebenen mit $f_u(x_1, x_2) = 0$ und $f_u(x_1, x_2) = 1$.

- Neuron v mit $m_v = 3$ Eingängen und $\theta_v = 1$

Linear trennbare Funktionen

Zwei Mengen $A, B \subseteq \mathbb{R}^n$ heißen genau dann linear trennbar, wenn eine lineare Funktion $g : \mathbb{R}^n \to \mathbb{R}$ mit $g(x_1, \ldots, x_n) = a_0 + \sum_{i=1}^{n} a_i x_1$ existiert, so dass

- für alle $(x_1, \ldots, x_n) \in A$ gilt $g(x_1, \ldots, x_n) > 0$
- für alle $(x_1, \ldots, x_n) \in B$ gilt $g(x_1, \ldots, x_n) < 0$

(eindeutig beschreiben durch $n+1$-Tupel $(a_0, a_1, \ldots, a_n)$)

Eine Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt genau dann linear trennbar, wenn die Mengen $f^{-1}(0)$ und $f^{-1}(1)$ linear trennbar sind.

Beispiele: $\vee, \wedge, \neg x_1, x_1 \to x_2, x_1 \wedge \neg x_2$

Die Boolesche Funktion `XOR` ist nicht linear trennbar.

McCulloch-Pitts-Neuron mit Hemmung

McCulloch-Pitts-Neuron u mit Hemmung:

Eingabewerte:	$x = (x_1, \ldots, x_{m_u}) \in \{0,1\}^{m_u}$	erregend
	$y = (y_1, \ldots, y_{m'_u}) \in \{0,1\}^{m'_u}$	hemmend
Schwellwert:	$\theta_u \in \mathbb{R}$	
Ausgabe:	$f(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) \in \{0,1\}$	

Parameter eines McCulloch-Pitts-Neurons u (mit Hemmung):

- m_u: Anzahl der erregenden Eingänge
- m'_u: Anzahl der hemmenden Eingänge
- θ_u: Schwellwert

Funktionen bei hemmenden Eingängen

Eingangsfunktion des Neurons u: $I_u : \{0,1\}^{m_u+m'_u} \to \mathbb{R} \times \mathbb{R}$

$$I_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) = \left(\sum_{i=1}^{m_u} x_i, \sum_{i=1}^{m'_u} y_i \right)$$

(Summe aller erregenden Eingänge des Neurons u,
Summe aller hemmenden Eingänge des Neurons u)
Aktivierungsfunktion des Neurons u (abhängig von θ_u):
$A_u : \mathbb{R} \times (\mathbb{R} \times \mathbb{R}) \to \{0,1\}$

$$A_u(\theta_u, (x,y)) = \begin{cases} 1 & \text{falls } x \geq \theta_u \text{ und } y \leq 0 \\ 0 & \text{sonst} \end{cases}$$

(Stufenfunktion)
Ausgabefunktion des Neurons u: O_u: $\{0,1\} \to \{0,1\}$ mit

$$O_u(v) = v$$

(Identität)

Berechnung bei hemmenden Eingängen

Gesamtfunktion des Neurons u

$$f_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) = O_u(A_u(\theta_u, I_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u})))$$

Jedes McCulloch-Pitts-Neuron u mit m_u erregenden Eingängen, m'_u hemmenden Eingängen und Schwellwert θ_u repräsentiert die Boolesche Funktion $f_u : \{0,1\}^{m_u+m'_u} \to \{0,1\}$:

$$f_u(x_1, \ldots, x_{m_u}, y_1, \ldots, y_{m'_u}) = \begin{cases} 1 & \text{falls } \sum_{i=1}^{m_u} x_i \geq \theta_u \\ & \text{und } \sum_{i=1}^{m'_u} y_i \leq 0 \\ 0 & \text{sonst} \end{cases}$$

Beispiele mit Hemmung:

- elementare Boolesche Funktion: $\neg$
- komplexere Boolesche Funktionen, z.B.

  $x_1 \wedge \neg x_2$

  $\neg x_1 \wedge x_2 \wedge x_3$,

  $\neg(x_1 \vee \neg x_2 \vee \neg x_3)$

McCulloch-Pitts-Netze

McCulloch-Pitts-Netz:

gerichteter Graph mit

- McCulloch-Pitts-Neuronen als Ecken und
- gerichteten Kanten zwischen Neuronen
  zwei Arten: erregend, hemmend

Berechnung der Neuronen-Funktionen
(entsprechend Struktur des Netzes):

- parallel
- sequentiell
- rekursiv

McCulloch-Pitts-Netze

Ein-Schicht-McCulloch-Pitts-Netz

parallele Schaltung mehrerer
Mc-Culloch-Pitts-Neuronen
repräsentiert Boolesche Funktionen mit mehreren Ausgaben
Beispiel: Parallelschaltung von $x_1 \wedge \neg x_2$ und $\neg x_1 \wedge x_2$

Mehr-Schicht-McCulloch-Pitts-Netz

parallele und sequentielle Schaltung mehrerer
Mc-Culloch-Pitts-Neuronen
Beispiel: XOR

Analogie zu logischen Schaltkreisen

Jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ lässt sich durch ein McCulloch-Pitts-Netz berechnen.

McCulloch-Pitts-Netz mit zwei Schichten genügt
(analog DNF, CNF in Aussagenlogik)

Modifikationen von McCulloch-Pitts-Neuronen

- Durch Vervielfachung eines Einganges erhöht sich seine Wirkung (sein Gewicht).
- Vervielfachung (absolut) hemmender Eingänge ändert die berechnete Funktion nicht.
- relative Hemmung:
  hemmende Eingänge verhindern das Feuern der Zelle nicht völlig, sondern erschweren es (erhöhen den Schwellwert, negatives Gewicht).
- Absolute Hemmung lässt sich durch relative Hemmung mit großer Schwellwerterhöhung (auf Anzahl aller erregenden Eingänge $+1$) simulieren.
- Durch Einführung von Gewichten wird Trennung in hemmende und erregende Eingänge überflüssig.

Parameter künstlicher Neuronen

verschiedene künstliche Neuronenmodelle unterscheiden sich in:

- Anzahl Typen der Ein- und Ausgabewerte,
- zulässige Gewichte an den Eingangskanten,
- Eingabe-, Ausgabe- und Aktivierungsfunktion

Jedes Neuron mit m Eingängen repräsentiert eine Funktion von m Eingabewerten

Schwellwertneuronen

Idee: gewichtete Eingänge

- zur Modellierung der Stärke der synaptischen Bindung
- ermöglichen Lernen durch Änderung der Gewichte

Mathematisches Modell:
Schwellwertneuron (Perzeptron)

Eingabewerte:	$x = (x_1, \ldots, x_m) \in \{0,1\}^m$	
Eingangsgewichte:	$w = (w_1, \ldots, w_m) \in \mathbb{R}^m$	
Schwellwert:	$\theta \in \mathbb{R}$	
Ausgabe:	$a(x_1, \ldots, x_m) \in \{0,1\}$	Aktivität

Parameter eines Schwellwertneuronens u:

- m_u: Anzahl der (erregenden) Eingänge
- $(w_1, \ldots, w_{m_u}) \in \mathbb{R}^{m_u}$: Eingangsgewichte
- θ_u: Schwellwert

Schwellwertneuronen: Funktionen

Eingangsfunktion des Neurons u (abhängig von $(w_1, \ldots, w_{m_u})$):
I_u: $\mathbb{R}^{m_u} \times \{0,1\}^{m_u} \to \mathbb{R}$ mit

$$I_u(w_1, \ldots, w_{m_u}, x_1, \ldots, x_{m_u}) = \sum_{i=1}^{m_u} w_i x_i$$

(gewichtete Summe aller Eingänge des Neurons u)
Aktivierungsfunktion des Neurons u (abhängig von θ_u):
A_u: $\mathbb{R} \times \mathbb{R} \to \{0,1\}$ mit

$$A_u(\theta_u, v) = \begin{cases} 1 & \text{falls } v \geq \theta_u \\ 0 & \text{sonst} \end{cases}$$

(Stufenfunktion)
Ausgabefunktion des Neurons u: O_u: $\{0,1\} \to \{0,1\}$ mit

$$O_u(v) = v$$

(Identität)

Schwellwertneuronen: Berechnung

vom Neuron u berechnete Funktion: f_u: $\{0,1\}^{m_u} \to \{0,1\}$ mit

$$\begin{aligned} f_u(x_1,\ldots,x_{m_u}) &= O_u(A_u(\theta_u, I_u(w_1,\ldots,w_{m_u},x_1,\ldots,x_{m_u}))) \\ &= \begin{cases} 1 & \text{falls } \langle w,x\rangle \geq \theta_u \\ 0 & \text{sonst} \end{cases} \end{aligned}$$

Wiederholung:
$\sum_{i=1}^{n} w_i x_i = \langle w,x\rangle$ Skalarprodukt
der Vektoren $w = (w_1,\ldots,w_n)$ und $x = (x_1,\ldots,x_n)$

Jedes Schwellwertneuron u mit m_u Eingängen repräsentiert eine Boolesche Funktion $f_u : \{0,1\}^{m_u} \to \{0,1\}$

Auch mit Schwellwertneuronen lassen sich nur linear trennbare Boolesche Funktionen berechnen (XOR nicht).
Beispiele: $\vee, \wedge, \to$, $((x_1 \wedge (x_3 \vee \neg x_2) \vee (\neg x_2 \wedge x_3)$

Schwellwertneuronen: geometrische Interpretation

Jedes Schwellwertneuron u mit m_u Eingängen teilt die Menge $\{0,1\}^{m_u}$ der Eingabevektoren (Punkte im $\mathbb{R}^{m_u}$) in zwei Teilmengen (Teilräume des R^{m_u}):

$$\begin{aligned} f_u^{-1}(1) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 1\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \langle w,x\rangle \geq \theta_u\} \end{aligned}$$

und

$$\begin{aligned} f_u^{-1}(0) &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid f(x_1,\ldots,x_{m_u}) = 0\} \\ &= \{(x_1,\ldots,x_{m_u}) \in \{0,1\}^{m_u} \mid \langle w,x\rangle < \theta_u\} \end{aligned}$$

Grenze: durch $\langle w,x\rangle = \theta_u$ beschriebene (m_u-1)-dimensionale Hyperebene (Teilraum)
(parallele Schnitte)

Schwellwert als Gewicht (Bias-Neuronen)

Neuron mit Schwellwert θ
Hinzufügen eines zusätzlichen Eingangs x_0 (bias neuron)
mit Wert $x_0 = 1$ (konstant)

Gewicht des Einganges x_0: $w_0 = -\theta$

$$\sum_{i=1}^{n} w_i x_i \geq \theta \quad \text{gdw.} \quad \sum_{i=1}^{n} w_i x_i - \theta \geq 0$$

$$\text{gdw.} \quad \sum_{i=0}^{n} w_i x_i \geq 0$$

Überwachtes Lernen einzelner Schwellwertneuronenn

Aufgabe: Konstruktion eines Schwellwertneurons zur Berechnung einer Booleschen Funktion $f : \{0,1\}^m \to \{0,1\}$

Trainingsmenge: Menge T von Paaren (x, t) aus

- Eingabevektoren $x \in \{0,1\}^m$ und
- Funktionswerten $t = f(x) \in \{0,1\}$

(Werte der Funktion f an Stützstellen)

Struktur des Schwellwertneuronens: Schwellwertneuron mit $m+1$ Eingängen (bias x_0) und Eingangsgewichten $(w_0, \ldots, w_m) \in \mathbb{R}^{m+1}$

Idee: automatisches Lernen der Funktion durch (wiederholte) Änderung der Gewichte

Lernziel: Gewichte $(w'_0, \ldots, w'_m) \in \mathbb{R}^{m+1}$, so dass das Schwellwertneuron die Funktion f berechnet (Korrektheit an Stützstellen)

Δ-Regel

Idee: Lernen aus Fehlern (und deren Korrektur)

Delta-Regel:

$$\forall i \in \{0, \ldots, m\} : w_i' = w_i + \Delta w_i \quad \text{mit} \quad \Delta w_i = \eta x_i (t - y)$$

- Trainingswert t
- vom Netz berechneter Wert y
- Lernrate $\eta \in \mathbb{R}$ (Grad der Verstärkung der Verbindung)

korrigierendes Lernen,
(falls x_i aktiv und $y \neq t$)

Beispiel: $\neg, \wedge, \rightarrow$

Δ-Lernverfahren für Schwellwertneuronen

- Beginn mit zufälligen Eingangsgewichten $(w_0, \ldots, w_n) \in \mathbb{R}^m$ (Schwellwert als Gewicht),
- die folgenden Schritte so oft wiederholen, bis der Fehler verschwindet (oder hinreichend klein ist):
  1. Bestimmung der Schwellwertneuron-Ausgabe y für Trainingspaar (x, t)
  2. Bestimmung des Fehlers $t - y$ der tatsächlichen zur gewünschten Ausgabe vom Trainingsziel t (als Funktion $e(w_0, \ldots, w_m)$ von den aktuellen Gewichten $w_0, \ldots, w_m$),
  3. Bestimmung geeigneter Gewichtsänderungen Δw_i
  4. Zuordnung der neuen Gewichte $w_i' = w_i + \Delta w_i$ zur Verringerung des (zukünftigen) Fehlers $(e(w_0', \ldots, w_n') < e(w_0, \ldots, w_n))$

Online-Lernen und Batch-Lernen

Lernen durch schrittweise

1. Berechnung des Fehlers
2. Berechnung der notwendigen Gewichtsänderungen
3. Änderung der Gewichte

Verfahren nach Zeitpunkt der Gewichtsänderung:

Online-Lernen Berechnung von Fehler und Gewichtsdifferenzen für jedes Trainingsmuster,
Änderung der Gewichte sofort für jedes Trainingpaar

Batch-Lernen (Lernen in Epochen)
Epoche: Berechnung für jedes Paar der Trainingsmenge
Berechnung von Fehler und Gewichtsdifferenzen für die gesamte Trainingsmenge (z.B. Summe über alle Trainingpaare)
Änderung der Gewichte erst nach einer ganzen Epoche

Konvergenz des Lernverfahrens

Konvergenzsatz:
Für jede Trainingsmenge

$$T \subseteq \{(x^{(i)}, t^{(i)}) \mid \forall i \in \{1, \ldots, n\} : x^{(i)} \in \{0,1\}^m\} \wedge t^{(i)} \in \{0,1\}\},$$

für welche die Mengen
$T_0 = \{x \mid (x,0) \in T\}$ und $T_1 = \{x \mid (x,1) \in T\}$
linear trennbar sind,
terminieren sowohl Online- als auch Batch-Lernen eines Schwellwertneuronens (passender Struktur) nach endlich vielen Schritten.

Die vom so trainierten Schwellwertneuron berechnete Funktion trennt die Mengen T_0 und T_1 voneinander.

Netze aus Schwellwertneuronen

Ein-Schicht-Schwellwertneuronen-Netz

parallele Schaltung mehrerer Schwellwertneuronen
repräsentiert Boolesche Funktionen mit mehreren Ausgaben
Beispiel: Parallelschaltung von $x_1 \wedge x_2$ und $\neg x_1 \wedge \neg x_2$

Mehr-Schicht-Schwellwertneuronen-Netz

parallele und sequentielle Schaltung mehrerer Schwellwertneuronen

Jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ lässt sich durch ein Schwellwertneuronen-Netz berechnen.

Schwellwertneuronen-Netz mit zwei Schichten genügt
(analog DNF, CNF in Aussagenlogik)

Netze aus Schwellwertneuronen

Ein-Schicht-Schwellwertneuronen-Netz

parallele Schaltung mehrerer Schwellwertneuronen
repräsentiert Boolesche Funktionen mit mehreren Ausgaben
Beispiel: Parallelschaltung von $x_1 \wedge x_2$ und $\neg x_1 \wedge \neg x_2$

Mehr-Schicht-Schwellwertneuronen-Netz

parallele und sequentielle Schaltung mehrerer Schwellwertneuronen

Jede Boolesche Funktion $f : \{0,1\}^n \to \{0,1\}$ lässt sich durch ein Schwellwertneuronen-Netz berechnen.

Schwellwertneuronen-Netz mit zwei Schichten genügt
(analog DNF, CNF in Aussagenlogik)

Feed-Forward-Netze (FFN)

- $V = \bigcup_{k=1}^{n} V_k$ mit $\forall i < j \in \{1, \ldots, n\} : V_i \cap V_j = \emptyset$
  Zerlegung der Menge der Neuronen in n disjunkte Schichten
- Menge der Eingangsneuronen: V_1 (je ein Eingang)
- Menge der Ausgangsneuronen: V_n (je ein Ausgang)
- Neuronen aller anderen Schichten heißen versteckte Neuronen
- $E \subseteq \bigcup_{k=1}^{n-1} V_k \times V_{k+1}$
  nur vorwärtsgerichtete Kanten zwischen benachbarten Schichten
- Gewichte bilden $m \times m$-Matrix (mit $m =$ Anzahl aller Neuronen)
- für FFN besteht die Gewichtsmatrix aus unabhängigen Blöcken
  Blöcke sind die Gewichtsmatrizen zwischen den Schichten

FFN als Berechnungsmodell:

- parallele Berechnung (in den Neuronen einer Schicht)
- sequentielle Berechnung (in miteinander verbundenen Neuronen benachbarter Schichten)

Perzeptron (historisch)

1958 Frank Rosenblatt, Idee: Modell der Netzhaut (Retina)
Aufbau des Perzeptrons:

1. Schicht (Eingabeschicht) : Menge S von Stimulus-Zellen (Verteilung)
2. Schicht (Mittelschicht) : Menge A von Assoziations-Zellen (Vorverarbeitung)
3. Schicht (Perzeptron-Schicht) : Menge R von Response-Zellen Muster-Assoziator aus Schwellwertneuronen (eigentliche Verarbeitung)

Verbindungen:

- zufällig zwischen Neuronen der Eingabeschicht und Neuronen der Mittelschicht
  feste Gewichte (zufällig)
- von jedem Neuron der Mittelschicht zu jedem Neuron der Ausgabeschicht
  trainierbare Gewichte

Jedes Ausgabeneuron teilt die Eingabemuster in zwei Klassen (akzeptierte und nicht-akzeptierte)

Ein-Schicht-FFN

- Abstraktion von der Eingabeschicht im historischen Perzeptron-Modell
- nur Perzeptron-Schicht (Muster-Assoziator)
- Parallele Berechnung mehrerer künstlicher Neuronen (hier Schwellwertneuronen)

Eingänge:	$(x_1, \ldots, x_m) \in \{0,1\}^m$
Ausgänge:	$(y_1, \ldots, y_n) \in \{0,1\}^n$
Gewichtsmatrix	$W \in \mathbb{R}^{m \times n}$

Gesamtberechnung des Ein-Schicht-FFN $f : \{0,1\}^m \to \{0,1\}^n$ des Neurons mit gewichteter Summe als Aktivierungsfunktion:
$f(x_1, \ldots, x_m) = (y_1, \ldots, y_n)$ mit $\forall k \in \{1, \ldots, n\}$:

$$y_k = \begin{cases} 1 & \text{falls } \sum_{i=1}^{m} = x_i w_{ij} \geq 0 \\ 0 & \text{sonst} \end{cases}$$

(Matrixmultiplikation)

Ein-Schicht-FFN: Training mit Δ-Regel

überwachtes Lernen

Trainingsmenge: Menge von Paaren (x, t) aus

- Eingabevektoren $x \in \{0,1\}^m$ und
- gewünschten Ausgabeverktoren $t \in \{0,1\}^n$

Lernen mit Delta-Regel für Ein-Schicht-FFN:

- Beginn mit zufälligen Eingangsgewichten $w_{ij} \in \mathbb{R}$,
- für jede Eingabe der Trainingsmenge (x, t):
  1. Netz berechnet die Ausgabe $y = xW$,
  2. Zuordnung neuer Gewichte w'_{ij} durch Delta-Regel:

$$w'_{ij} = w_{ij} + \Delta(w_{ij}) \quad \text{mit} \quad \Delta(w_{ij}) = \eta x_i (t_j - y_j)$$

- wiederholen, bis der Fehler klein genug ist.

Das Lernverfahren mit Delta-Regel konvergiert für

- jede linear trennbare Boolesche Funktion f und
- hinreichend kleine Lernquote η

in endliche vielen Schritten zu einem Ein-Schicht-FFN, welche die Funktion f berechnet.

Künstliche Neuronen mit reellen Ein- und Ausgängen

Parameter:

Eingänge:	$x_1, \ldots, x_m \in \mathbb{R}^m$
Eingangsgewichte	$w_1, \ldots, w_m \in \mathbb{R}^m$
Ausgang:	$f(\langle x, w \rangle) \in \mathbb{R}$

- Eingangsfunktion $I : \mathbb{R}^m \to \mathbb{R}$
- Aktivierungsfunktion $A : \mathbb{R} \to \mathbb{R}$
- Ausgabefunktion $O : \mathbb{R} \to \mathbb{R}$

Gesamtberechnung $f : \mathbb{R}^m \to \mathbb{R}$ des Neurons:

$$f(x_1, \ldots, x_m) = O(A(I(x_1, \ldots, x_m)))$$

Klassifikation durch Ein-Schicht-FFN

Klassifikation:
Zerlegung einer Menge M von Werten in (paarweise disjunkte) Klassen $\{C_1, \ldots, C_n\}$, welche die Wertemenge vollständig überdecken

$$\bigcup_{i=1}^{n} C_i = M \quad (\forall i \neq j : C_i \cap C_j = \emptyset)$$

Klassifikation des $\mathbb{R}^m$ durch KNN:

- Eingänge $(x_1, \ldots, x_m) \in \mathbb{R}^m$
- Ausgänge $(y_1, \ldots, y_n) \in \{0, 1\}^n$
  für jede Klasse C_i ein Ausgabeneuron y_i
  Ausgang $y_i = 1$ gdw. Eingabe $(x_1, \ldots, x_m) \in C_i$

überwachtes Training des Ein-Schicht-FFN:

- zufällige Startgewichte
- schrittweise Modifikation der Gewichte zur Verringerung des Fehlers

Ein-Schicht-FFN erkennt nur linear trennbare Klassen

Problem: Wie trainiert man Mehrschicht-FFN?

Auswahl durch Mehrschicht-FFN – Beispiel

Beispiel: Auswahl aller Punkte im Einheitsquadrat

$$y = \begin{cases} 1 & \text{falls } 0 \leq x_1 \leq 1 \wedge 0 \leq x_2 \leq 1 \\ 0 & \text{sonst} \end{cases}$$

durch das 2-Schicht-FFN mit

- Eingängen x_1, x_2 und x_0 (bias)
- Ausgang y
- versteckten Neuronen $z_1, \ldots, z_4$ und z_0 (bias)
- Gewichte der ersten Schicht (zwischen (x_0, x_1, x_2) und $(z_1, \ldots, z_4)$:

$$W_1 = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

  z_1 feuert gdw. $x_1 \leq 1$, z_2 feuert gdw. $x_1 \geq 0$
  z_3 feuert gdw. $x_2 \leq 1$, z_4 feuert gdw. $x_2 \geq 0$

- Gewichte der zweiten Schicht (zwischen $(z_0, \ldots, z_4)$ und y):

$$W_2 = (-7/2, 1, 1, 1, 1)^T$$

Gesamtmatrix des FFN – Beispiel

	x_0	x_1	x_2	z_0	z_1	z_2	z_3	z_4	y
x_0	0	0	0	0	1	0	1	0	0
x_1	0	0	0	0	1	-1	0	0	0
x_2	0	0	0	0	0	0	1	-1	0
z_0	0	0	0	0	0	0	0	0	$-7/2$
z_1	0	0	0	0	0	0	0	0	1
z_2	0	0	0	0	0	0	0	0	1
z_3	0	0	0	0	0	0	0	0	1
z_4	0	0	0	0	0	0	0	0	1
y	0	0	0	0	0	0	0	0	0

Mehr-Schicht-FFN mit linearer Aktivierung

Netzeingänge:	$(x_1, \ldots, x_{k_0}) \in \mathbb{R}^m$	
Netzausgänge:	$(y_1, \ldots, y_{k_l}) \in \mathbb{R}^n$	
Neuronen (l Schichten):	$(z_1^0, \ldots, z_{k_0}^0) \in \mathbb{R}^{k_1}$	(Eingabeneuronen)
	$\vdots$	(versteckte Neuronen)
	$(z_1^l, \ldots, z_{k_l}^l) \in \mathbb{R}^{k_l}$	(Ausgabeneuronen)
Gewichtsmatrizen	$W^{(j)} \in \mathbb{R}^{k_j \times k_{j+1}}$ für jedes $j \in \{0, \ldots, l-1\}$	

lineare Aktivierungsfunktion $I : \mathbb{R} \to \mathbb{R}$ mit $I(x) = mx$

Ausgabe des Neurons z_i^j in Schicht j:

$$f(z_1^{j-1}, \ldots, z_{k_{j-1}}^{j-1}) = O(A(I(x_1, \ldots, x_{k_{j-1}}))) = m\left(\sum_{l=1}^{k_{j-1}} w_{li}^{(j)} z_l^{(j-1)}\right)$$

Netzausgabe:

$$f(x_1, \ldots, x_m) = m'(x_1, \ldots, x_m)W^{(0)} \cdots W^{(l-1)} = m'(x_1, \ldots, x_m)W$$

mit $W = W^{(0)} \cdots W^{(l-1)}$ (Matrixmultiplikation)

Jede Funktion, die von einem Mehr-Schicht-FFN mit linearer Aktivierung berechnet wird, kann also auch durch ein Ein-Schicht-FFN mit linearer Aktivierung berechnet werden.

Approximation von Funktionen

gegeben: Menge von Trainingspaaren $\{(x^{(1)}, t^{(1)}), \ldots, (x^{(k)}, t^{(k)})\}$
k Stützstellen und Werte an diesen Stützstellen
(z.B. Messwerte)

Ziel:
Konstruktion eines KNN zur Approximation dieser Funktion durch

- lineare Funktionen
- Stufenfunktionen
- komplexere Funktionen

Quadratischer Fehler

Approximation einer Menge von Trainingspaaren
(Funktionswerte an Stützstellen)
durch Funktion gegebenen Typs (z.B. linear)

- Trainingsmenge liefert Stützstellen:

$$(x_{k1}, \ldots, x_{kn}, t_k)_{k \in \{1,\ldots,m\}}$$

- approximierende Funktion $f : \mathbb{R}^n \to \mathbb{R}$
- Fehler an der Stützstelle $(x_{k1}, \ldots, x_{kn})$:

$$t_k - f(x_{k1}, \ldots, x_{kn})$$

- quadratischer Fehler an der Stützstelle $(x_{k1}, \ldots, x_{kn})$:

$$E_k = (t_k - f(x_{k1}, \ldots, x_{kn}))^2$$

- quadratischer Gesamtfehler (Summe über alle Trainingspaare / Stützstellen):

$$E = \sum_{k=1}^{m} (t_k - f(x_{k1}, \ldots, x_{kn}))^2$$

Trainingsziel: Minimierung des quadratischen Fehlers

Beispiel

Bestimmung der Parameter m, n einer Geraden $y = f(x) = mx + n$ aus einer Menge gegebener (ungenauer) Trainingspaare (x, t), z.B.:

$$\{(1, 10), (2, 7), (4, 5), (5, 1)\}$$

(ganz einfaches) Ein-Schicht-FFN:

- ein Eingang x_1, ein Bias-Neuron x_0
- ein Ausgangsneuron y
- Gewichte: $w_0 = n, w_1 = m$

Funktionen des Ausgabeneurons y:

- Eingangsfunktion I: gewichtete Summe $nx_0 + mx_1 = mx_1 + n$
- Aktivierungsfunktion A: Identität (linear)
- Ausgangsfunktion O: Identität

Dieses Netz berechnet die Funktion

$$f(x) = O(A(I(x_1))) = I(x_1) = mx_1 + n$$

Ermittlung der Parameter m, n durch Training des Netzes (Δ-Regel)

Methode der kleinsten Quadrate

direkte Berechnung mit Hilfe der partiellen Ableitungen nach m und n

$$E = \sum_{k=1}^{l} (t_k - f(x_k))^2 = \sum_{k=1}^{l} (t_k - mx_k - n)^2$$

partielle Ableitungen nach m und n:

$$\begin{aligned}
\frac{\partial E}{\partial m} &= \sum_{k=1}^{l} (-2)\,(t_k - mx_k - n)\,x_k \\
&= -2\left(\sum_{k=1}^{l} t_k x_k - m\sum_{k=1}^{l} x_k^2 - n\sum_{k=1}^{l} x_k\right) \\
\frac{\partial E}{\partial n} &= \sum_{k=1}^{l} (-2)\,(t_k - mx_k - n) \\
&= -2\left(\sum_{k=1}^{l} t_k - m\sum_{k=1}^{l} x_k - nl\right)
\end{aligned}$$

Bestimmung der Parameter

Im Minimum von f sind alle partiellen Ableitungen 0.
Das ergibt ein lineares Gleichungssystem für m und n:

$$\begin{aligned} \sum_{k=1}^{l} t_k x_k - m \sum_{k=1}^{l} x_k^2 - n \sum_{k=1}^{l} x_k &= 0 \\ \sum_{k=1}^{l} t_k - m \sum_{k=1}^{l} x_k - ln &= 0 \end{aligned}$$

mit den Lösungen

$$\begin{aligned} n &= \frac{\sum_{k=1}^{l} t_k - m \sum_{k=1}^{l} x_k}{l} \\ m &= \frac{l \sum_{k=1}^{l} t_k x_k - \left(\sum_{k=1}^{l} t_k\right)\left(\sum_{k=1}^{l} x_k\right)}{\sum_{k=1}^{l} x_k^2 - \left(\sum_{k=1}^{l} x_k\right)^2} \end{aligned}$$

im Beispiel $m = -2, n = 47/4$

Berechnung der Gewichts-Verschiebungen

Ziel:
Minimierung des Fehlers durch schrittweise Verschiebung des Gewichtsvektors

Methode: Gradientenabstiegsverfahren
Verschiebung des Gewichtsvektors in Richtung des steilsten Abstieges (entgegen dem steilsten Anstieg) der Fehlerfunktion (als Funktion der Gewichte)

steilster Anstieg: Gradient (partielle Ableitungen)

Gradientenabstiegsverfahren führt oft, aber nicht immer zu einem geeigneten (globalen) Minimum der Fehlerkurve,
endet mitunter in lokalem Minimum

Voraussetzung: Fehlerfunktion ist differenzierbar

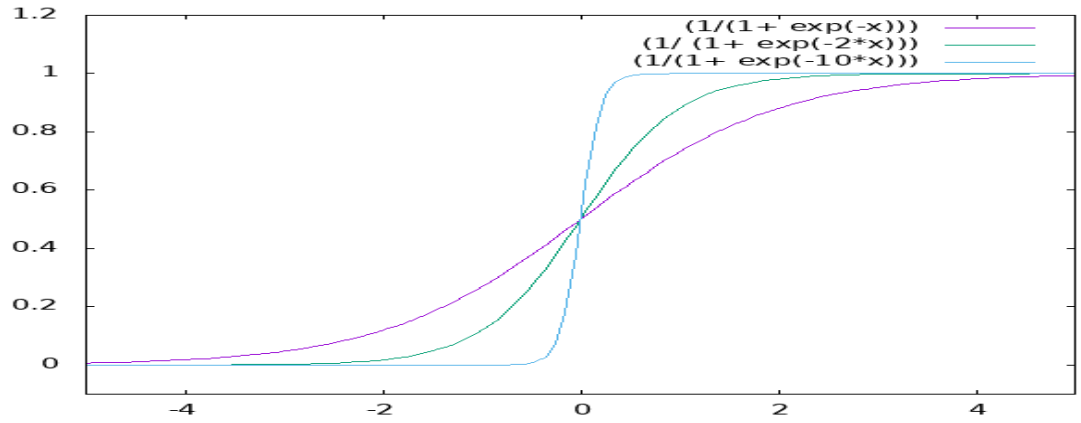
zur Anwendung in KNN: differenzierbare Aktivierungsfunktion

Sigmoide Aktivierungsfunktion

differenzierbare Approximation der Stufenfunktion:

sigmoide Funktion

$$f(x) = \frac{1}{1+e^{-x}} \quad \text{mit Parameter } c > 0: \quad f(x) = \frac{1}{1+e^{-cx}}$$



\+ überall differenzierbar
Ableitung im Punkt x:

$$s'(x) = \left(\frac{1}{1+e^{-x}}\right)' = \frac{e^{-x}}{(1+e^{-x})^2} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right) = s(x)(1-s(x))$$

in jedem Punkt eindeutige Abstiegsrichtung

\- erreicht die Werte 0 und 1 nie,
Toleranzbereiche notwendig, so entstehen Ungenauigkeiten

Lineare Aktivierungsfunktionen und ReLU

lineare Aktivierung: $\forall x \in \mathbb{R} : A(x) = mx + n$

+ einfach (schnell) zu berechnen
  überall differenzierbar
  Ableitung: konstant m
  in jedem Punkt $x > 0$ eindeutige Abstiegsrichtung

ReLU(Rectified Linear Units): $\forall x \in \mathbb{R} : A(x) = \max(0, x)$

+ einfach (schnell) zu berechnen
  fast überall differenzierbar
  Ableitung: Stufenfunktion, 0 bei $x < 0$, 1 bei $x > 0$,
  in jedem Punkt $x > 0$ eindeutige Abstiegsrichtung
- Problem: Ableitung nicht definiert bei $x = 0$
  (aber praktisch nicht relevant)

Beispiel

(ganz einfaches) Ein-Schicht-FF-Netz: ein Neuron mit

- einem Eingang $x \in \mathbb{R}$,
- einem Gewicht $w \in \mathbb{R}$,
- Eingabefunktion $I(x) = wx$ (gewichtete Summe)
- verschiedene Aktivierungsfunktionen $A : \mathbb{R} \to \mathbb{R}$
- Ausgabefunktion: $O(x) = x$

berechnet eine Funktion $f : \mathbb{R} \to \mathbb{R}$ mit

$$y = f(x) = O(A(I(x))) = A(wx)$$

quadratischer Fehler für ein Trainingspaar (x, t):

$$E(w) = (t - y)^2 = (t - f(x))^2 = (t - A(wx))^2$$

Ableitung der Fehlerfunktion nach dem Eingangsgewicht w:

$$\frac{\partial E(w)}{\partial w} = E'(w) = 2(t - A(wx))A'(wx) = 2(t - A(wx))xA'(w)$$

Beispiel mit identischer Aktivierungsfunktion

$$y = f(x) = O(A(I(x))) = A(wx) = wx$$

quadratischer Fehler:

$$E(w) = (t - y)^2 = (t - A(wx))^2 = (t - wx)^2$$

Ableitung nach w:

$$\frac{\partial E(w)}{\partial w} = -2(t - wx)x = -2(t - y)x$$

Gewichtsänderung:

$$\Delta w = -\eta' \frac{\partial E(w)}{\partial w} = \eta(t - y)x \quad (\Delta\text{-Regel})$$

Beispiel mit sigmoider Aktivierungsfunktion

$$y = f(x) = O(A(I(x))) = A(wx) = \frac{1}{1 + e^{-wx}}$$

quadratischer Fehler:

$$E(w) = (t - y)^2 = (t - A(wx))^2 = \left(t - \frac{1}{1 + e^{-wx}}\right)^2$$

Ableitung nach w:

$$\frac{\partial E(w)}{\partial w} = -2(t - A(wx))A'(wx) = -2(t - y)y(1 - y)x$$

Gewichtsänderung:

$$\Delta w = -\eta \frac{\partial E(w)}{\partial w} = \eta(t - y)y(1 - y)x$$

(Backpropagation-Regel für die Ausgabeschicht)

Allgemeines Ein-Schicht-FF-Netz

Ein-Schicht-FF-Netz mit

- Eingängen $x \in \mathbb{R}^m$,
- Ausgängen $y \in \mathbb{R}^n$,
- Gewichtsmatrix $W \in \mathbb{R}^{m \times n}$
  (Gewicht w_{ij} zwischen Eingang i und Ausgang j),
- Eingangsfunktion $I(x) = \sum_{i=1}^{m} x_i w_{ij}$
  (gewichtete Summe der Eingänge am Neuron j, Skalarprodukt von x mit Spalte j der Gewichtsmatrix W)
- Ausgangsfunktion $O(x) = x$ (Identität)

berechnet eine Funktion $f : \mathbb{R}^m \to \mathbb{R}^n$ mit

$$y_j = f(x_1, \ldots, x_m) = O(A(I(x_1, \ldots, x_m))) = A\left(\sum_{i=1}^{m} x_i w_{ij}\right)$$

quadratischer Fehler für ein Trainingspaar $(x, t) \in \mathbb{R}^m \times \mathbb{R}^n$:

$$E = \sum_{j=1}^{n} (t_j - y_j)^2 = \sum_{j=1}^{n} \left(t_j - A\left(\sum_{i=1}^{m} x_i w_{ij}\right)\right)^2$$

Gewichtsänderungen

quadratischer Fehler für ein Trainingspaar $(x, t) \in \mathbb{R}^m \times \mathbb{R}^n$:

$$E = \sum_{j=1}^{n} (t_j - y_j)^2 = \sum_{j=1}^{n} \left(t_j - A \left(\sum_{i=1}^{m} x_i w_{ij} \right) \right)^2$$

Ableitung nach w_{ij}:

$$\begin{aligned} \frac{\partial E}{\partial w_{ij}} &= \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial A(I(x_1, \ldots, x_m))}{\partial I(x_1, \ldots, x_m)} \frac{\partial I(x_1, \ldots, x_m)}{\partial w_{ij}} \\ &= (t_j - y_j) \frac{\partial A(I(x_1, \ldots, x_m))}{\partial I(x_1, \ldots, x_m)} x_i \end{aligned}$$

Gewichtsänderungen:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta (t_j - y_j) \frac{\partial A \left(\sum_{i=1}^{m} x_i w_{ij} \right)}{\partial w_{ij}}$$

Beispiele

identische Aktivierung $A(x) = x$

$$\frac{\partial A\left(\sum_{i=1}^{m} x_i w_{ij}\right)}{\partial w_{ij}} = \frac{\partial \sum_{i=1}^{m} x_i w_{ij}}{\partial w_{ij}} = x_i$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = \eta(t_j - y_j)x_i \quad \text{(Delta-Regel)}$$

sigmoide Aktivierung $A(x) = \frac{1}{1+e^{-x}}$

$$\frac{\partial A\left(\sum_{i=1}^{m} x_i w_{ij}\right)}{\partial w_{ij}} = y_j(1 - y_j)x_i$$

$$\Delta w_{ij} = -\eta' \frac{\partial E}{\partial w_{ij}} = \eta(t_j - y_j)y_j(1 - y_j)x_i$$

Mehrschicht-FFN

- Eingabeschicht x
- versteckte Schichten $z^{(1)}, \ldots, z^{(n)}$
- Ausgabeschicht y

gewichtete Verbindungen zwischen

- x und $z^{(1)}$
- für alle $i \in \{0, \ldots, n_i\}$ zwischen $z^{(i)}$ und $z^{(i+1)}$
- $z^{(n)}$ und y

Darstellung der Gewichte zwischen benachbarten Schichten als Matrizen
(nur relevante Blöcke der gesamten Gewichtsmatrix)

Backpropagation in FFN

(Bryson, Ho 1969, Rummelhard, McClelland 1986)

Ziel: Geeignete Modifikation aller Gewichte im FFN zur Verrringerung des Gesamtfehlers

Idee:

- Betrachte jedes Gewicht w_{uv} als Eingangsgewicht des Teilnetzes zwischen Neuron v und Netz-Ausgängen
- Netzeingabe in dieses Teilnetz ist $w_{uv}o_u$ mit Netzausgabe o_u des Neurons u
- partielle Ableitung $\frac{\partial E}{\partial w_{uv}} = o_u\delta_v$ mit Fehleranteil $\delta_v = o_v(1-o_v)\sum_p w_{vp}\delta_p$, wobei p über alle direkten Nachfolger von v läuft

Backpropagation-Training

in jedem Schritt 2 Durchläufe des FFN:

Vorwärts-Schritt: Berechnung der Netzausgabe
Speichern der Netzausgabe o_u in jedem Neuron u
Speichern der Ableitung der Netzausgabe $o_u(1 - o_u)$ in jedem Neuron u

Rückwärts-Schritt: Berechnung des Fehleranteils δ_u jedes Neurons aus den Fehleranteilen aller Nachfolger-Neuronen
$\delta_u = o_u(1 - o_u) \sum_p w_{vu}\delta_p$,
Speichern der Fehleranteile δ_u in jedem Neuron u

danach Anpassung aller Gewichte um $\Delta w_{uv} = -\eta o_u \delta_v$

Zwei-Schicht-Feed-Forward-Netz – Beispiel

(ganz einfaches) Zwei-Schicht-Feed-Forward-Netz:

- Eingabe: ein Neuron x
  keine gewichteten Eingänge
  Eingangs-, Aktivierungs- und Ausgangsfunktion: Identität
- versteckte Schicht: ein Neuron h
  ein gewichteter Eingang (von x, Gewicht w_{xh})
  Eingangsfunktion: gewichtete Summe, hier nur $w_{xh}x$
  Aktivierungsfunktion: sigmoid $A_h(v) = \frac{1}{1+e^{-v}}$
  Ausgangsfunktion: Identität
- Ausgabe: ein Neuron y
  ein gewichteter Eingang (von h, Gewicht w_{hy})
  Eingangsfunktion: gewichtete Summe, hier nur $w_{hy}h$
  Aktivierungsfunktion: sigmoid
  Ausgangsfunktion: Identität

Netz berechnet die Funktion $f : \mathbb{R} \to \mathbb{R}$ mit

$$f(x) = f_y(f_h(x)) = O_y(A_y(I_y(O_h(A_h(I_h(x))) = A_y(w_{hy}A_h(w_{xh}x))$$

(Verkettung von Funktionen)

Backpropagation-Methode – Beispiel

Backpropagation-Schritte für ein Trainingspaar (x, t):

1. Vorwärts-Schritt: Funktionskomposition
   schichtweise Berechnung der Neuronen-Ein- und -Ausgaben
   - Berechnung der Ein- und Ausgaben jedes Neurons aus der Eingabe x
     $o_h = O_h(A_h(I_h(x))) = \frac{1}{1+e^{-w_{xh}x}}$,
     $o_y = O_y(A_y(I_y(h))) = \frac{1}{1+e^{-w_{hy}o_h}}$
   - Berechnung der Netzausgabe $y = o_y$
   - Berechnung des Fehlers $E = (y - t)^2$
2. Rückwärts-Schritt: Multiplikation
   schichtweise Berechnung der anteiligen Fehler δ_h, δ_y nach Gradientenabstiegsverfahren
   - Ausgabeschicht y:
     $\delta_y = -\frac{\partial E}{\partial A_y} = (t - o_y)A'_y = (t - o_y)o_y(1 - o_y)$
   - versteckte Schicht h: $\delta_h = \delta_y w_{hy} o_h(1 - o_h)$
3. Aktualisierung der Gewichte
   $\Delta w_{xh} = \eta\delta_h x$, $\Delta w_{hy} = \eta\delta_y o_y$

Allgemeine Mehr-Schicht-Feed-Forward-Netze

FFN mit k Schichten $s \in \{0, \ldots, k\}$ zu je n_s Neuronen und Gewichten $w_{ij}^{(s)}$ zwischen Ausgang des Neurons i der Schicht $s-1$ und Eingang des Neurons j der Schicht s
k Gewichtsmatrizen $W^s \in \mathbb{R}^{n_{s-1}} \times \mathbb{R}^{n_s}$

Verallgemeinerung der Backpropagation-Methode auf

- Parallelität (mehrere Neuronen je Schicht)
  - Vorwärts-Schritt: Addition mehrerer Eingaben
  - Rückwärts-Schritt: partielle Ableitungen
- Kantengewichte: Multiplikation (beide Richtungen)
- mehrere versteckte Schichten:
  mehrere Vorwärts- und Rückwärtsschritte

Backpropagation-Lernen allgemein

- Instanziierung aller Gewichte mit kleinen zufälligen Werten
- BP-Verfahren für eine Epoche:
  - BP-Verfahren für jedes Traingsmuster (x, t):
    - Vorwärtsschritt (Ausgabe-Berechnung):
      für jede Schicht s (Beginn bei Eingabeschicht):
      Berechnung der Vektoren $z^{(s)} = I(y^{(s-1)})$ und
      $y^{(s)} = A(z^{(s)}) = A(I(y^{(s-1)}))$ für jedes Neuron der Schicht s
    - Rückwärtsschritt (Gewichtsdifferenzen):
      für die Ausgabeschicht k:
      Berechnung des Vektors $d^{(k)} = (t - y^{(k)})y^{(k)}(1 - y^{(k)})$
      für jede Schicht s (Beginn bei letzter versteckter Schicht
      $k - 1$):
      Berechnung des Vektors $d_j^{(s)} = y_j^s(1 - y_j^{(s)}) \sum_{m=1}^{n^{(s+1)}} d_m^{(s+1)} w_{mj}$
      für jedes Neuron j der Schicht s
    - Aktualisierung aller Gewichte: $w_{ij}^{(s)} := w_{ij}^{(s)} + \eta d_j^{(s)} y_i^{(s)}$

    danach weiter mit nächstem Trainingsmuster (x', t')

  danach weiter mit nächster Epoche
- Ende, falls erreichte Änderung des Fehlers klein (unter einer Schranke)

Backpropagation-Lernen mit Trägheit

zur Vermeidung von

- Oszillationen in „Schluchten“ und
- Abbremsen auf Plateaus

$$w_{ij}^{(s)} := (1 + \alpha) w_{ij}^{(s)} + \eta d_j^{(s)} y_i^{(s)}$$

mit Trägheit α

Anwendung von FFN mit Backpropagation

KNN zur Muster-Klassifikation

Klassifikation von Eingabemustern, z.B.

- optische Zeichenerkennung
  (z.B. Buchstaben, abstrahiert von Schriftart)
- Erkennung akustischer Signale (z.B. Stimmen)
- englische Ausspracheregeln (NETTALK)
- Datenkompression (Eingabe = Ausgabe, Code in der versteckten Schicht)
- Vertrauenswürdigkeit von Bankkunden (Risikoklassen)
- Vorhersage (Wetter, Aktienkurse)
- bisher: Boolesche Funktionen
  (Klassifikation von Eingabevektoren nach Ausgabe-Wahrheitswerten)

Qualität von BP-Netzen

gute Generalisierung:
KNN klassifiziert die meisten neuen Eingabemuster einer Testdatenmenge (nicht aus der Trainingsmenge) richtig
abstrahiert von kleinen Abweichungen
abhängig von

- Netzarchitektur (nicht zu viele versteckte Neuronen)
- Auswahl der Trainingsmenge

Problem:
übertrainierte Netze kennen die Trainingsmenge „auswendig“