Was bisher geschah

- biologisches Vorbild neuronaler Netze und Lernvorgänge darin
- künstliche Neuronen (mit binären Ein- und Ausgängen):
  - McCulloch-Pitts-Neuron (ohne Eingangsgewichte)
  - Schwellwertneuron (mit Eingangsgewichten)
- Feed-Forward-Netze
  gerichteter Graph mit Kantengewichten (Matrix)
  (parallele und sequentielle Berechnung)
- Verwendung künstlicher neuronaler Netze:
  - Lernphase (aufwendig, aber nur einmal auszuführen)
  - Einsatzphase (schnell, wird oft ausgeführt)
- Lernverfahren:
  - überwacht
    - korrigierend, z.B. durch Δ-Regel
    - bestärkend
  - unüberwacht
- überwachtes Lernen eines Schwellwertneurones durch schrittweise Änderung der Gewichte (Δ-Regel)

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

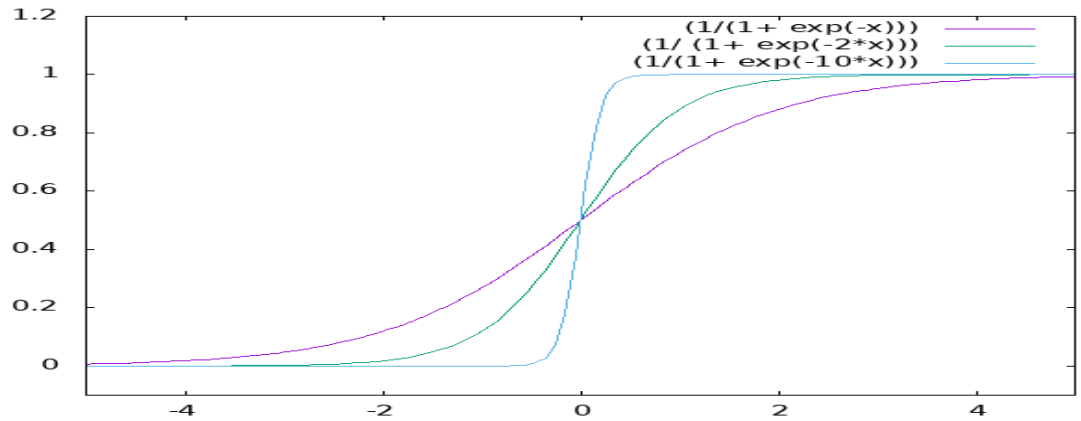
zur Anwendung in KNN: differenzierbare Aktivierungsfunktion

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:

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

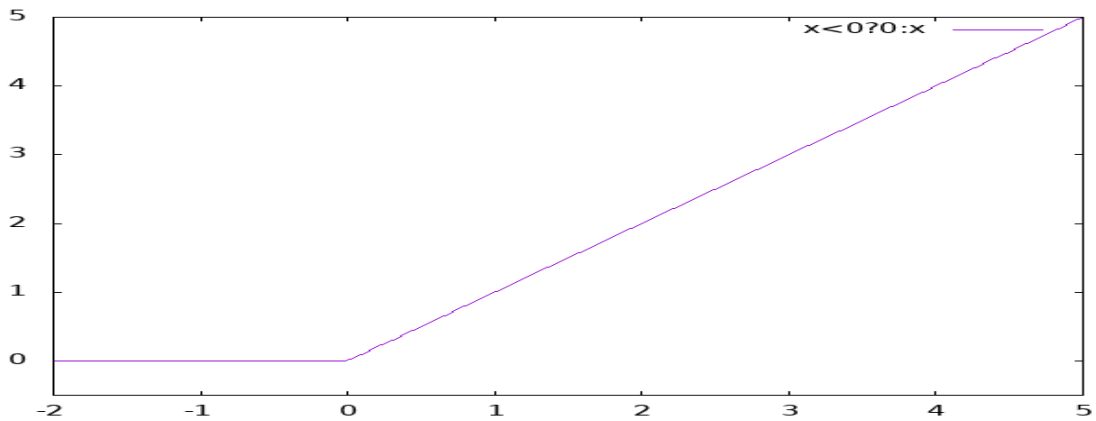
in jedem Punkt eindeutige Abstiegsrichtung

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

Aktivierungsfunktion 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“

Rekurrente Netze: Motivation

Ziel: Nachnutzung von Informationen aus vorangegangenen Schritten, z.B. zur

- Repräsentation zeitlicher Folgen von Mustern
- Zeitreihenanalyse und -voraussage
- Erkennung von Sätzen (Grammatik)
- Verarbeitung von Mustern variabler Länger (betrachtet als Sequenzen)

mögliche Ansätze

- gleitendes Zeitfenster:
  FFN mit n Eingabeneuronen
  Eingabemuster enthält Informationen aus n vorangegangenen Schritten
  Nachteil: beschränkte Breite des Zeitfensters
  - Erkennen „entfernter“ Abhängigkeiten schwierig
  - viele Eingabeneuronen nötig
- rekurrente KNN

Wiederholung: allgemeine KNN

Netzstruktur (Topologie):
gerichteter Graph $G = (V, E)$ mit

- endliche Menge $V = \{v_1, \ldots, v_n\}$ von Knoten (Neuronen)
  evtl. einige als Eingabe- bzw. Ausgabeneuronen gekennzeichnet (nicht notwendig)
- Menge $E \subseteq V \times V$ von (gewichteten) Kanten

eine Gewichtsmatrix $R^{V \times V}$ für alle möglichen Verbindungen zwischen Neuronen

Rekurrente KNN

Netze mit Kanten zwischen beliebigen Neuronen

erlaubt Nachnutzung von Ausgaben aus vorangegangenen Schritten

Repräsentation zeitlicher Folgen von Mustern

Idee:
aktuelle Ausgaben als Eingaben im nächsten Schritt nutzen

„Kurzzeitgedächtnis“

Netzstruktur:

- analog Feed-Forward-Netz
- zusätzliche Neuronen und Kanten für Rückkopplung
  (Informationsspeicherung bis zum folgenden Schritt)

Beispiel

- zwei McCulloch-Pitts-Neuronen u, v
- Eingang $x \in \{0, 1\}$
- Ausgang $y \in \{0, 1\}$
- erregende Kanten: $(x, u), (x, v), (u, u), (u, v), (v, y)$
- hemmende Kanten $(v, v), (v, u)$
- Schwellwerte $\theta_u = 1$, $\theta_v = 2$

Zustand rekurrenter Netze

Zustand eines neuronalen Netzes (zeitveränderlich)
Aktivierung aller Neuronen:
Zuordnung S: Neuron $\rightarrow \mathbb{R}$

(evtl. genügen Kontextneuronen)

Übersetzung in Zustandsübergangssysteme
(endliche Automaten)

Zu jedem NFA existiert ein rekurrentes Netz mit McCulloch-Pitts-Neuronen, welches dieselben Zustandsübergänge simuliert.

Mathematisches Modell: Rekursion

Wiederholung: KNN als Berechnungsmodell

FFN als Berechnungsmodell:

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

rekurrentes Netz als Berechnungsmodell:

- mehrmalige Nacheinanderausführung einer Funktion (ohne Abbruchbedingung)
  Berechnung einer rekursiven Funktion
  (Fixpunkt)

„Entwirrung“ rekurrenter Netze

Idee:

- Verarbeitung von Eingaben zu Ausgaben eines Neurons kostet einen Zeitschritt
- für jeden Zeitschritt eine Kopie aller Neuronen und Kanten dazwischen,
- Ersetzung der Rückwärtskanten durch Vorwärtskanten zur nächsten Kopie.

In diesem expandierten Netz ist Lernen der Vorwärtskanten durch Backpropagation-Verfahren möglich:

- Durchlauf jeder Netz-Kopie ist ein Zeitschritt,
- Lernen durch Backpropagation des entwirrten KNN (Backpropagation through time)

Jordan-Netze

Idee: Nachnutzung der Netzausgaben
Netz-Topologie:

- Feed-Forward-Netz mit trainierbaren Vorwärtskanten,
- für jedes Ausgabeneuron ein zusätzliches Kontextneuron in der Eingabeschicht
  (zur Speicherung der Netzausgaben)
  Aktivierungsfunktion: Identität
- zusätzliche Verbindungen von jedem Neuron der Ausgabeschicht zu seinem Kontextneuron mit
  festen Gewichten λ (meist $\lambda = 1$),
  Speicherung der Ausgaben
- evtl. direkte Verbindungen von jedem Kontextneuron zu sich selbst mit festem Gewicht γ
  (zur weiteren Speicherung der Netzausgaben)
- zusätzliche Verbindungen von jedem Kontextneuron zu jedem Neuron der ersten versteckten Schicht mit
  trainierbaren Gewichten,
  (zur Verwendung der gespeicherten Ausgabe im Folgeschritt)

Jordan-Netze: Berechnung

Beispiel: Eingang = gewichtete Summe, Aktivierung = Identität,
$x(t)$ – Netzeingabe zum Zeitpunkt t
$S(t)$ – Zustand (Aktivierung der Kontextneuronen) zum Zeitpunkt t

Ausgabe: $y(t) = f(x(t), S(t))$

(Zustands-)Übergangsfunktion: $S(t+1) = g(x(t), S(t))$

Zustand des Netzes nach mehreren Schritten (Schritt für gesamtes Netz), beginnend im Startzustand S_0

$$
\begin{aligned}
S(t) &= \begin{cases} S_0 & \text{falls } t = 1 \\ \gamma S(t-1) + \lambda y(t-1) & \text{falls } t > 1 \end{cases} \\
&= \gamma^{t-1} S_0 + \lambda \sum_{n=1}^{t-1} \gamma^{n-1} y(t-n)
\end{aligned}
$$

Spezialfall $S_0 = 0$ und $\lambda = 1$: $\quad S(t) = \sum_{n=1}^{t-1} \gamma^{n-1} y(t-n)$

exponentiell gewichtete Summe aller bisherigen Netzausgaben

$\gamma \in [0, 1]$ steuert „Erinnerungsvermögen“ des Netzes

Elman-Netze

Idee: Nachnutzung der Aktivierung der versteckten Neuronen

Netz-Topologie:

- Feed-Forward-Netz (z.B. SRN 3-Schicht-FFN)
- für jedes versteckte Neuron ein zusätzliches Kontextneuron in der vorigen Schicht
  (zur Speicherung der Aktivierung)
  Aktivierungsfunktion: Identität
- zusätzliche Verbindungen von jedem versteckten Neuron zu seinem Kontextneuron mit festem Gewicht 1
  Speicherung der Aktivierung aller versteckten Neuronen
- zusätzliche Verbindungen von jedem Kontextneuron zu jedem Neuron der Schicht des Originalneurons mit trainierbaren Gewichten,
  (zur Verwendung der gespeicherten Aktivierung im Folgeschritt)