Neuro-Info Notizen

Markus Klemm.net

WS 2016/2017

Inhaltsverzeichnis

1 Hebbsche Lernregel **1**
1.1 Fälle 1
1.2 Lernrate 2

2 Neural Gas **2**
2.1 Algorithmus 2
2.1.1 Was macht ein Neuron 3
2.1.2 Trainingsalgorithmus 3
2.1.3 Self-organizing Feature Maps 4

3 Multilayer-Perceptron (MLP) **5**

4 Grundprinzipien des SNG **7**

1 Hebbsche Lernregel

$$w_{xy}^{\text{neu}} = w_{xy}^{\text{alt}} + \Delta w_{xy} \tag{1}$$

$$\Delta w_{xy} = \eta \cdot o_x \cdot a_y \tag{2}$$

Notation: w_{xy} ... Gewicht zwischen Neuron x (Sender) und Neuro y (Empfänger)
Δw_{xy} ... Gewichtsänderung
η ... Lernrate, sinnvollerweise $[0; 1]$
o_x ... ≥ 0 Ausgabe des Neurons x
a_y ... ≥ 0 Aktivierung des Neurons y
$h(,)g(,)$ ... beliebige Funktionen

1.1 Fälle

1. $o_x \vee a_y \to 0 \Rightarrow \Delta w_{xy} = 0$ keine Gewichtsänderung Verbindung wird nicht verstärkt
2. $o_x > 0 \wedge a_y > 0 \Rightarrow \Delta w_{xy} > 0$ Gewicht wird adaptiert $\Rightarrow$ Verbindung prägt sich aus

1.2 Lernrate

1. Konstante Wahl
   $\eta \uparrow \ldots$ große Sprünge auf den Fehlergebrige/-Funktion $\curvearrowright$ „enge“ Minima werden Übersprungen $\Rightarrow$ Optima werden verpasst
2. $\eta \downarrow \ldots$ langsame konvergent $\rightarrow$ lernen dauert lange $\eta = \eta(t)$ verändert über Zeit, beginnen mit großen Werten und enden mit Kleinen

Wo sind jetzt eigentlich die Vektoren Gewicht $\underline{w} \in \mathbb{R}^n$ Eingangsgrößen $\underline{x} \in \mathbb{R}^n$
Problemstellung I: unsupervised learning $\rightarrow$ unüberwacht heißt keine Trainingsgrößen. Um diese Daten weiter verarbeiten zu können entweder

- alle Daten speichern und „mitschleifen“
- Repräsentaten wählen

Was ist ein Neuron? Prof. Böhme: einzelne Gewichte in Summenfunktionen, Ausgabe als nicht lineare Funktion $\rightarrow \underline{w} = (w_1, w_2, w_n)^T$ Kohonen: nur das Gewicht !1elf im selbem Raum wie Eingaberaum

2 Neural Gas

2.1 Algorithmus

(Trainings-)

- Initialisierung: Starte mit zufälligen Neurozentren $w_k \in \mathbb{R}^n (\forall k \leq K)$
- Anlegen eines Eingabemusters $x(t) \in \mathbb{R}^n \quad (t \leq T)$
- Abstandsmessung: Für jedes Neuron w_k bestimme $\|x(t) - W_k\|_2$
- Bestimmen des Best-Matching Neurons w_b
  $\forall i: \ \|x(t) - w_i\| \geq \|x(t) - w_b\|$
- Erstelles einer Liste von Neuronen sortiert nach ihrer Distanz zum Datenpunkt $x(t)$ Ziel: Nährere Neuronen stärker beeinflusst
- Adaption der Neuronen (Gewichte) $\forall i$
  $w_i^{\text{neu}} = w_i^{\text{alt}} + \Delta w_i$
  $\Delta w_i = \eta \cdot h_i \cdot (x(t) - w_i)$
- Wiederholung(ab nach Initialisierung), einmal für alle Datenpunkte sog. Trainingsepoche und dann noch bis zufrieden bzw. konvergent

Nachteile:

- Sortieren der Liste $\rightarrow$ Rechenaufwand
- „Löcher“ in der Abbildung, da kein Zusammenhalt der Neuronen
- Nachbarschaften können sich jederzeit ändern

2.1.1 Was macht ein Neuron

$$z_i = w_{13}x_i + w_{23}y_2 - \Theta = 0$$

(Geradengleichung) in der Form: $y = mx + n$
$x_2 = -\frac{w_{13}}{w_{23}} \cdot x_1 + \frac{\Theta}{w_{23}}$

Beispiel Neuron 3 soll logische UND Abbildung lernen

Trainingsdaten:

x_1	x_2	t
0	0	0
0	1	0
1	0	0
1	1	1

Ziel: Punkt der Klasse 1 $\vec{x} = \begin{pmatrix}1\\1\end{pmatrix}$ von Punkten der Klasse 0 zu separieren $\rightarrow$ linear separierbares Problem, durch lineare Diskriminanzfunktion lösbar.

$\forall$ Punkte $\vec{x} = \begin{pmatrix}x_1\\x_2\end{pmatrix}$, die auf der Geraden G liegen, gilt $z_3 = 0$. Wir vereinbaren nicht-lineare Transferfunktion $y_3 = f(z_3) = \begin{cases} 1 & \text{für} \quad z_3 \geq 0 \\ 0 & \text{für} \quad z_3 < 0 \end{cases}$
Gesucht NW-Parameter (Gewichte und Schwellen) mit denen Neuron 3 Gerade G bildet w_{13}, w_{23}, Θ

$x_2 = -\frac{w_{13}}{w_{23}} \cdot x_1 + \frac{\Theta}{w_{23}}$ aus Skizze ablesbar $n = 1,5, m = -1$. Anstieg von G wird durch das Verhältnis $\frac{w_{13}}{w_{23}}$ bestimmt: Annahme $w_{13} = 1 \curvearrowright w_{23} = 1, \Theta = 1,5$

Probe: für $\vec{x}^1 = \begin{pmatrix}0\\1.5\end{pmatrix}, \vec{x}^2 = \begin{pmatrix}1,5\\0\end{pmatrix}$
$z_3(\vec{x}^1) = 1 \cdot 0 + 1 \cdot 1,5 - 1,5 = 0; z_3(\vec{x}^2) = 1 \cdot 1,5 + 1 \cdot 0 - 1,5 = 0$
$\vec{x}^3 = \begin{pmatrix}1\\1\end{pmatrix} : z_3 = 1 \cdot 1 + 1 \cdot 1 - 1,5 = 0,5 \curvearrowright y_3 = 1$
$\vec{x}^4 = \begin{pmatrix}0\\1\end{pmatrix} : z_3 = 1 \cdot 0 + 1 \cdot 1 - 1,5 = -0,5 \curvearrowright y_3 = 0$ Schwelle von G beworkt eine Verschiebung von G aus dem Koordinatenursprung oder anders: bewirkt eine Verschiebung des „Arbeitsbereiches“ des Neurons entlang der z-Achse in der Funktion $y = f(z)$
Wenn Θ ansteigt, muss das Skalarprodukt $\vec{W}^T \circ \vec{X}$ größer werden, um diese zu erreichen bzw zu überschreiten. $Z = \vec{W}^T \circ \vec{X} - \Theta$

- Schritt 1: Initialisiere die Parameter zufällig, z.B. aus Intervall $[-1; 1]$
- Schritt 2: nehme Trainingsdaten und wende Training-Alg. an

2.1.2 Trainingsalgorithmus

Ziel Parameter so adaptieren, dass Differenz (das Delta) zwischen vorgegebener Ausgabe $\vec{t}$ und der tatsächlichen NW-Ausgabe $\vec{y}$ minimal wird.

Einfachste Variante: Delta-Lernregel

$$\Delta w_{ij} = \eta \cdot x_i \cdot (t_j - y_j)$$

- iterative Anwendung der Delta-Lernregel erfolgt für das einschichtige Perceptron (Rumelhard/McClelland 1965)
- Perceptron liefert linearen Klassifikator
- kann keine nicht linear trennbaren Bereiche repräsentieren, z.B. XOR-Abbildung

Problem Wie trainiert man ein mehrschichtiges Perceptron? z.B. 2 schichtiges Perceptron Eingang;Hidden-Schicht;Ausgabeschicht
Idee: den Fehler am NW-Ausgang zur Adaption aller Parameter nutzen

- Fehlerfunktion E als Funktionen jedes einzelnen Parameters notieren
- Minimum dieser Funktions mittels iterativem Gradientenabstiegs suchen
- Fehlerfunktion $E_p(\vec{X}^p, \vec{W}) = \frac{1}{2} \sum_{k=1}^{K} (t_k - y_k)^2$ in dieser Funktion ist die Abhängigkeit von jedem einzelnen Parameter durch eine Verkettung der Funktion $z(x)$ und $y(z)$ enthalten
  $E(w_{jk}) : E = f(y_k); y_k = f(z_k); z_k = f(w_{jk}); E(y_k(z_k(w_{jk})))$
  $E(w_{ij}) : E = f(y_k); y_k = f(z_k); z_k = f(y_j); y_j = f(z_j); z_j = f(w_{ij}); E(y_k(z_k(y_j(z_j(w_{ij})))))$
- um stetige Differenzierbarkeit zu gewährleisten ersetzen wir $\begin{cases} 1 & \text{für} \quad z \geq 0 \\ 0 & \text{für} \quad z < 0 \end{cases}$ durch $y = \frac{1}{1+e^{-z}}$

2.1.3 Self-organizing Feature Maps

- Initialisierung:
  - Starte Netz it zufälligem Neuronenzentren $w_k \in \mathbb{R}^N (\forall k \leq k)$
  - Def. Gitterstruktur
- $\forall t$: Anlegen des Eingabemusters $x(t) \in \mathbb{R}^N$
- Abstandsmessung: $\forall w_k$ bestimme $\|x(t) - w_k\|_2$
- Bestimmen des Best-Matching-Neurons w_b:
  $\forall i : \|x(t) - w_i\| \geq \|x(t) - w_b\|$
- Adaption der Neuronen $w_i^{\text{neu}} = w_i^{\text{alt}} + \Delta w_i$
  $\forall i : \Delta w_i = \eta \cdot h_{bi} \ (x(t) - w_i)$

3 Multilayer-Perceptron (MLP)

Topologie $y_{\text{bias}} = 1$! Für Neuronen 5 bis 9:

- Skalarproduktaktivierung
- $y_i = \frac{1}{1+e^{-z_i}}$

$$\vec{x} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \quad \vec{t} = \begin{pmatrix} t_8 \\ t_9 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Wir initialisieren alle PArameter mit 0! $\vec{w_5} = \vec{w_6} = \vec{w_7} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}; \vec{w_8} = \vec{w_9} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

z.B. $w_5 = \begin{pmatrix} w_{B5} \\ w_{15} \\ w_{25} \\ w_{35} \\ w_{45} \end{pmatrix}; \vec{w_8} = \begin{pmatrix} w_{B8} \\ w_{58} \\ w_{68} \\ w_{78} \end{pmatrix}; \eta = 0,1$

Aufgabe Ausgaben y_8 und y_9 für $\vec{x}$ bestimmen $\vec{x^*} = \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix}; \vec{y^\#} = \begin{pmatrix} 1 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix}$

Berechnung der Ausgabe der Hiddenschicht: $z_5 = \vec{W_5}^T \cdot \vec{X}^* = 0 \circlearrowright y_5 = \frac{1}{1+e^{-0}} = 0,5$
$z_6 = \vec{W_6}^T \cdot \vec{X}^* = 0 \circlearrowright y_6 = 0,5$
$z_7 = \vec{W_7}^T \cdot \vec{x^*} = 0 \circlearrowright y_7 = 0.5$

$$\vec{y^\#} = \begin{pmatrix} 1 \\ 0.5 \\ 0.5 \\ 0.5 \end{pmatrix}$$

$z_8 = \vec{W_8}^T \cdot \vec{y^\#} = 0 \circlearrowright y_8 = 0,5$
$z_6 = \vec{W_9}^T \cdot \vec{y^\#} = 0 \circlearrowright y_9 = 0,5$

Wir nutzen die Differenz zwischen $\vec{t}$ & $\begin{pmatrix} y_8 \\ y_9 \end{pmatrix}$ zur Adaption der Gewichte. Differenz/Fehlerfunktion $E = \underbrace{\frac{1}{2} \sum_i (t_i - y_i)^2}_{\text{am Ausgang}}$

$E(y(z(w)))$ für jeden einzelnen Parameter w von Hidden zur Ausgabeschicht
$E(y(z(y(z(w)))))\forall$ Parameter von Eingabe zur Hiddenschicht

Idee Gradientenabstieg

$$\Delta w = \eta \cdot (-\frac{\partial E}{\partial w})$$

$w_(t+1) = w(t) + \Delta w(t)$

Bestimmung von $\frac{\partial E}{\partial w}$:

für Gewichte Hidden $\mapsto$ Ausgang

$\frac{\partial E}{\partial w_{jk}} = \sum_k \frac{\partial t}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_k} \cdot \frac{z_k}{w_{jk}}$

$\frac{\partial E}{\partial y_k} : \frac{\partial \frac{1}{2} \sum_k (t_k - y_k)^2}{\partial y_k} = \frac{1}{2} \sum_k (t_k^2 - 2t_k y_k + y_k^2)|_{y_k} = \frac{1}{2} \sum_k (-2t_k + 2y_k) = \sum_k (y_k - t_k)$

$\frac{\partial y_k}{\partial z_k} : \frac{1}{1+e^{-z_b}}|_{z_k} = (1+e^{-z_k})^{-1}|_{z_k} = \frac{(-1)\cdot(-1)\cdot e^{-z_k}}{(1+e^{-z_k})\cdot(1+e^{-z_k})} = \frac{e^{-z_k}+1-1}{(1+e^{-z_k})(1+e^{-z_k})} = \underbrace{\frac{1}{1+e^{-z_k}}}_{y_k} \cdot \underbrace{\frac{e^{-z_k}+1-1}{1+e^{-z_k}}}_{(1-y_k)} =$

$y_k(1-y_k)$

$\frac{\partial z_k}{\partial w_{jk}} : y_1 \cdot w_{1k} + \cdots + y_j \cdot w_{jk} + \cdots + y_7 \cdot w_{Jk}|_{w_{jk}} = y_j$ (Kante zwischen 5 und 8) w_{jk} mit $j = 5, k = 8$ (Wichtig $i = [1; I], j = [1; J], k = [1; K]$)

$$\frac{\partial E}{\partial w_{jk}} = \underbrace{\sum_k (y_k - t_k)}_{\frac{\partial E}{\partial y_k}} \cdot \underbrace{y_k \cdot (1 - y_k)}_{F'(z_k) = \frac{\partial y_k}{\partial z_k}} \cdot \underbrace{y_j}_{\frac{\partial z_k}{\partial w_{jk}}}$$

$\forall$ Parameter von Eingabe zur Hiddenschicht

$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial y_j} \cdot \frac{\partial y_j}{\partial z_j} \cdot \frac{\partial z_j}{\partial w_{ij}} = \frac{\partial E(y_k(z_k(y_j(z_j(w_{ij})))))}{\partial w_{ij}}$

$$\frac{\partial E}{\partial w_{ij}} = \underbrace{\sum_k (y_k - t_k)}_{\frac{\partial E}{\partial y_k}} \cdot \underbrace{y_k(1 - y_k)}_{\frac{\partial y_k}{\partial z_k}} \cdot \underbrace{w_{jk}}_{\frac{\partial z_k}{\partial y_j}} \cdot \underbrace{y_j(1 - y_j)}_{\frac{\partial y_j}{\partial z_j}} \cdot \underbrace{y_i}_{\frac{\partial z_j}{\partial w_{ij}}}$$

$\frac{\partial (\frac{1}{2} \sum_k (t_k - y_k)^2)}{\partial y_k} = \frac{1}{2} \sum_k (t_k^2 - 2t_k y_k + y_k^2)|_{y_k} = \sum_k (y_k - t_k)$

$$\Delta w_{ij} = \underbrace{\left[\underbrace{\sum_k \underbrace{(t_k - y_k) \cdot y_k(1 - y_k)}_{\delta_k} w_{ij} y_j (1 - y_j)}_{\delta j} y_i \right]}_{(\frac{\partial E}{\partial w_{ij}})(-1)} \cdot \eta$$

$$\Delta w_{jk} = \underbrace{\left[\sum_k (t_k - y_k) y_k (1 - y_k) y_j \right]}_{(\frac{\partial E}{\partial w_{jk}})(-1)} \cdot \eta$$

$\forall$ Knoten einer (beliebigen) Hiddenschicht gilt: Fehler berechnet sich aus gewichteter Summe der Fehler der jeweils vorgelagerten Schicht und das Ganze multiplziert mit der Ableitung der Transferfunktion. ($y = f(z)$

[Btw Abstände $||x||_n := (\sum_i |x_i|)^{\frac{1}{n}}$]

4 Grundprinzipien des SNG

- Reiner Vektorquantisierer
- keine apriori-Festlegung der Anzahl der Referenzvektoren
- zusätzlich zum Distanrang der Referenzvektoren bzgl. des aktuellen Stimulus
  - Definition der Nachbarschaft mit Hilfe einer Graphenstruktur
  - Nachbar eines Knoten i: Sind alle Knoten, die mit i durch eine Kante verbunden sind
  - jeder Knoten verfügt über Zahähler z mit dessen Hilfe der Einfügeprozess gesteuert wird
  - Zählt wie oft Neuron Bm-N Wieiviele Daten das Neuron „gesehen“ hat
- jede Kante besitzt einen Alterparameter γ der das Löschen von Kanten steuert (und damit implizit das Löschen von Knoten)
  - Restrukturierung der Toplologie
  - Knoten die sehr lange nicht SBM-N (Second best matching neuron) oder in direkter Nachbarschaft waren, sterben aus
- Adaption der Referenzvektoren erfolgt bei jeder Musterpräsentation
- Adaption der Topologie erfolgt jeweils nach vorgegebner Anzahl von Trainingsparametern
- Bestandsaufnahme der bisherigen Verfahren
- Prototypen zur Datenrerpsäentation
- Klassengrenzen Varonoi-Resionen es fehlt: Klassenlabel für Prototypen Künnen wir das einfach hinzufügen