Multi-Layer Networks and Learning Algorithms

16.12.03

Referent: Alexander Perzylo

Betreuer: Martin Bauer

Überblick

- Multi-Layer Perceptron und Back-Propagation
- Hopfield Netze (Hebb-Regel)
- Kohonen Self-Organizing Maps

Mögliche Lernabläufe in NN

- Einführen/Löschen von Neuronen
- Entwicklung/Löschen von Verbindungen
- Modifikation der Verbindungs-Gewichtungen
- Modifikation der Neuronen-Schwellenwerte
- Modifikation der Aktivierungs-, Propagierungs- oder Ausgabefunktion

Arten von Lernverfahren

- Überwachtes Lernen (supervised)
  - Lernalgorithmus minimiert den Fehler zwischen berechnetem und gewünschtem Output
- Unüberwachtes Lernen (unsupervised)
  - Optimierung von Gewichten basierend auf Kriterien, die das Netz selber hat
- Bestärkendes Lernen (reinforcement)
  - Lernalgorithmus adaptiert Gewichte basierend auf Maximierung einer Belohnung (reward) bzw. Minimierung einer Bestrafung (punishment), die aufgrund des berechneten Outputs ausgesprochen wird

Back-Propagation Learning

- überwachtes Lernen
- für Multi-Layer Feedforward Netze
- Idee:
  - Präsentation eines Eingabevektors
  - Propagierung vorwärts durch das Netz
  - Bestimmung des Ausgabevektors
  - Ermittlung des Fehlermaßes für die Ausgabe
  - Rückschreitend schrittweise Ermittlung des Fehlermaßes der Neuronen der verdeckten Schicht(en)
  - Änderung der Gewichte der Neuronen der Ausgangsschicht und der verdeckten Schichten anhand ihrer Fehleranteile

Back Propagation: Pseudocode

function NEURAL-NETWORK-LEARNING(*examples*) **returns** *network*

network ← a network with randomly assigned weights
repeat
  for each *e* **in** *examples* **do**
    O ← NEURAL-NETWORK-OUTPUT(*network*, *e*)
    T ← the observed output values from *e*
    update the weights in *network* based on *e*, **O**, and **T**
  end
until all examples correctly predicted or stopping criterion is reached
return *network*

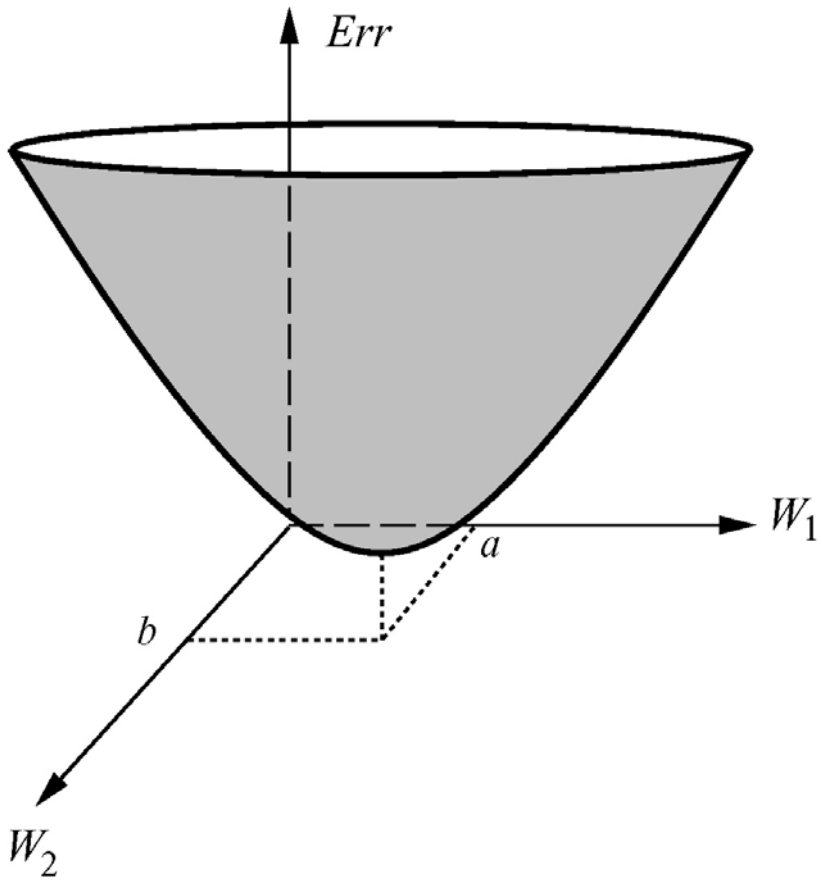
function BACK-PROP-UPDATE(*network, examples*, α) **returns** a network with modified weights

inputs: *network*, a multilayer network
examples, a set of input/output pairs
α, the learning rate

repeat
  for each *e* **in** *examples* **do**
    /* *Compute the output for this example* */
    $\mathbf{O} \leftarrow \text{RUN-NETWORK}(network, \mathbf{I}^e)$
    /* *Compute the error and* Δ *for units in the output layer* */
    $\mathbf{Err}^e \leftarrow \mathbf{T}^e - \mathbf{O}$
    /* *Update the weights leading to the output layer* */
    $W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times Err_i^e \times g'(in_i)$
    for each subsequent layer **in** *network* **do**
      /* *Compute the error at each node* */
      $\Delta_j \leftarrow g'(in_j) \sum_i W_{j,i} \Delta_i$
      /* *Update the weights leading into the layer* */
      $W_{k,j} \leftarrow W_{k,j} + \alpha \times I_k \times \Delta_j$
    end
  end
until *network* has converged
return *network*

Backpropagation: mathemat. Hintergrund

- Back Propagation ist Gradientenabstiegs-verfahren auf der Fehleroberfläche über der Menge aller Gewichtungen



Backpropagation: mathemat. Hintergrund

- Gewicht Update Regel in Ausgabeschicht:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \cdot a_j \cdot Err_i \cdot g'(in_i)$$

  - Wir definieren den Fehlerterm Δ :

$$\Delta_i = Err_i \cdot g'(in_i)$$

  - Daraus folgt die Regel:

$$W_{j,i} \leftarrow W_{j,i} + \alpha \cdot a_j \cdot \Delta_i$$

Backpropagation: mathemat. Hintergrund

- Fehlerterm Δ für Eingabeschicht:

$$\Delta_j = g'(in_j)\sum_i W_{j,i}\Delta_i$$

- Gewicht Update Regel für Eingabeschicht:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \cdot I_k \cdot \Delta_j$$

Backpropagation: mathemat. Hintergrund

- Wir verwenden als Fehlerfunktion:

$$E = \frac{1}{2}\sum_i (T_i - O_i)^2$$

- Für allgemeines 2 Schichten Netz gilt:

$$E(W) = \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i} a_j))^2$$

$$= \frac{1}{2}\sum_i (T_i - g(\sum_j W_{j,i} g(\sum_k W_{k,j} I_k)))^2$$

Backpropagation: mathemat. Hintergrund

- Nun leiten wir *E* nach $W_{j,i}$ ab:

$$\frac{\partial E}{\partial W_{j,i}} = -a_j (T_i - O_i) g'(\sum_j W_{j,i} a_j)$$

$$= -a_j (T_i - O_i) g'(in_i) = -a_j \Delta_i$$

- Ableitung von *E* nach $W_{k,j}$ ähnlich:

$$\frac{\partial E}{\partial W_{k,j}} = \ldots = -I_k \Delta_j$$

Back-Propagation

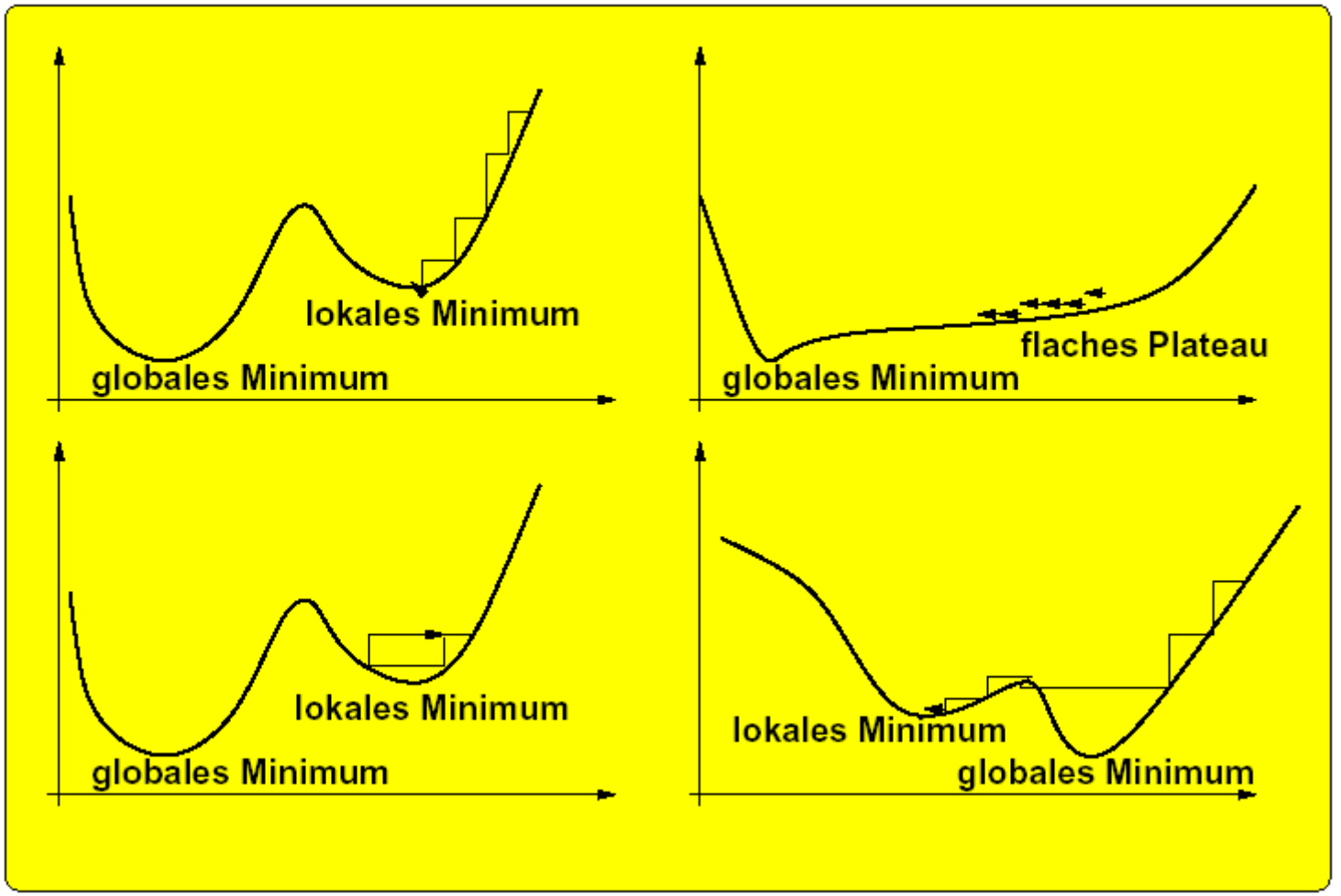
- Erinnerung:
  Ziel ist Minimierung des Fehlers
  => Um Gewicht Update Regeln zu erhalten:
  - Gewichte in entgegengesetzte Richtung des Gradienten der Fehlerfunktion bewegen

- Anmerkung: sigmoid-Funktion als Aktivierungsfunktion
  - Vorteil: $\frac{\partial sigmoid}{\partial x} = sigmoid(1 - sigmoid(x))$

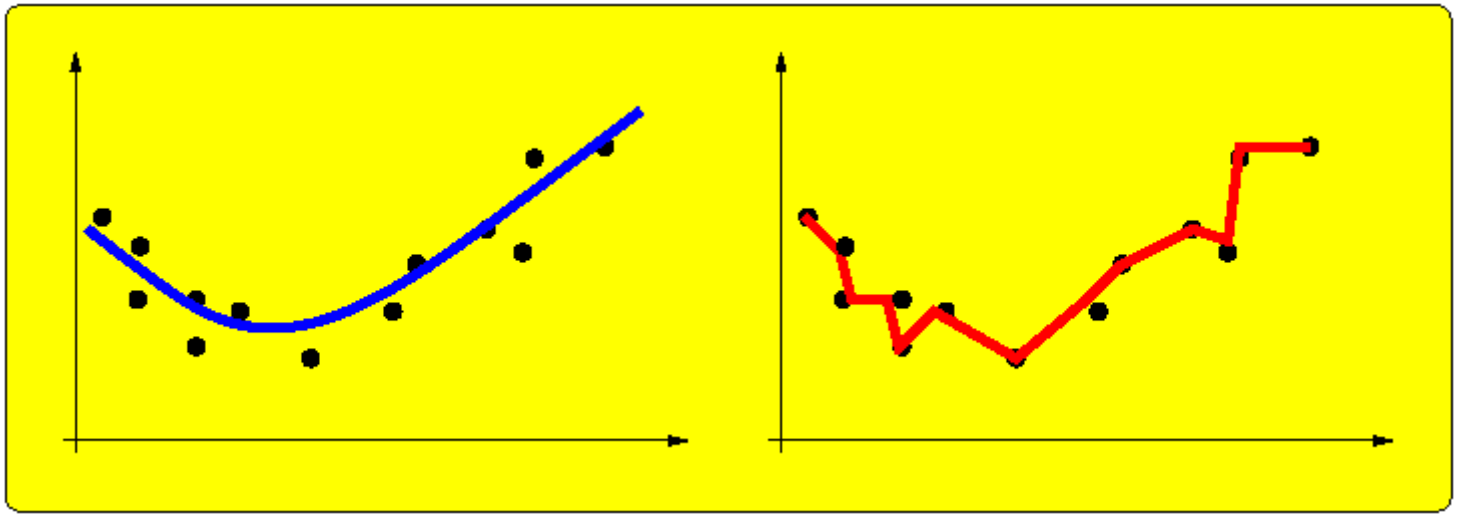
Back-Propagation

- Probleme:
  - Anzahl der Verdeckten Neuronen und der Lernrate vom Anwendungsgebiet abhängig
  - Konvergenz zu einer (optimalen) Lösung nicht immer gegeben
  - Nur lokales Minimum möglich
  - Optimale Minima können während des Lernens verlassen werden
  - Gefahr des Überlernens (Overfitting,Overshooting)
  - keine Begründung für ein Ergebnis (Blackbox)

Back-Propagation: Konvergenzprobleme



Back Propagation: Overfitting



Gegenmaßnahmen:

Zufällige Reihenfolge der Trainigsdaten und Validierung des Netzes mit anderen Beispielen als dem Trainingsset

Multi-Layer Network: Berechnungseffizienz

- Hängt von Zeit ab, die benötigt wird, das Netz auf ein Set von Beispielen zu trainieren
- Ein Durchlauf (Epoche) eines Trainingssets mit m Beispielen und w Gewichten hat eine Zeitkomplexität von O(mw)
- Im Worst-case: Anzahl Durchläufe = 2^n , n = Dimension des Eingabevektors
  - Variiert jedoch sehr stark in der Praxis

Multi-Layer Network: Anwendungsbeispiele

- (1987) NETtalk kann geschriebenen englischen Text vorlesen. Input besteht aus dem auszusprechenden Buchstaben und dem Vorgänger und 3 Nachfolgern
  - 80 Verdeckte Neuronen.
  - Output-Schicht: hoch, tief, betont, unbetont,...
  - Training: 1024 Worte. Nach 50 Epochen: 95% korrekt auf Trainingsmenge.
  - Test-Menge: Zu 78 % korrekt.

Multi-Layer Network: Anwendungsbeispiele

- (1989) handgeschriebene Postleitzahlen auf Briefumschlägen erkennen:
  - Eingabe: 16x16 Pixel-Array.
  - 3 Verdeckte Schichten, mit jeweils 768, 192, 30 Neuronen. 10 Ausgabeneuronen (0-9). Das ergibt 200000 Gewichte!
  - Trick: Neuronen in der ersten Schicht untersuchen nur eine 5 x 5 Teilmatrix: 768 = 12 x 64. Jede der 12 Gruppen benutzt gleiche Gewichte und ist für ein Merkmal zuständig => Insgesamt 9760 Gewichte
  - Training: 7300 Beispiele, Test-Menge: zu 99% korrekt erkannt

Multi-Layer Network: Anwendungsbeispiele

- Auto fahren: (1993) ALVINN (Autonomous Land Vehicle In a Neural Network)
  - Input: Stereo Farb-Kamera, Radar
  - Bild wird in ein 30 x 32 Pixel-Array umgewandelt (Eingabevektor)
  - Verdeckte Neuronen: Eine Schicht mit 5 Neuronen
  - Output-Schicht: 30 Neuronen (entspricht Lenk-Winkel)
  - Ausgewählt wird das Ausgabe-Neuron mit höchster Aktivierung
  - Training: Ein Mensch fährt 5 Minuten und das Netz „sieht zu“
  - Probleme: Mensch fährt zu gut, stark Helligkeitsabhängig (Wetter, Straßenbelag)

Hopfield Netz

- ist autoassoziativer Speicher
- Funktion: Mustervervollständigung / Musterklassifizierung
  - Zunächst: gewünschte Muster werden im Netz „abgespeichert“
  - neue Eingaben können vervollständigt bzw. klassifiziert werden

Hopfield Netz

- alle Neuronen miteinander verbunden
- Eingabe- zugleich Ausgabeschicht
  - Dimension n (n = #Neuronen)
  - ist bipolar (enthält Werte –1 und 1)
- Symmetrische Gewichte: $W_{j,i} = W_{i,j}$
- lernt mit Hebb-Regel: „Die synaptische Eigenschaft (Verstärken oder Hemmen) ändert sich proportional zum Produkt von prä- und postsynaptischer Aktivität“

Hopfield Netz

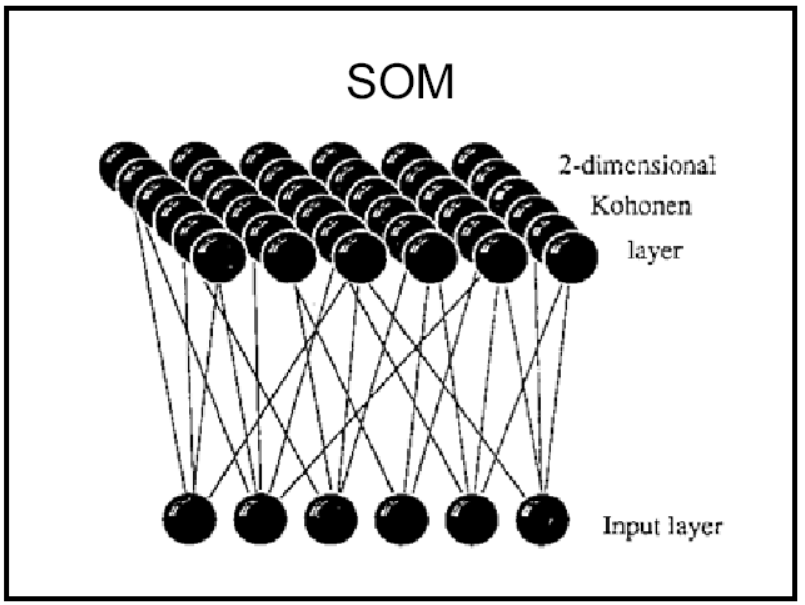
- „Merkvorgang“:
  - Startzustand = zu lernendes Muster
  - Modifikation der Gewichte:
    - Für alle vorhandenen Verbindungen:
      - Verstärkung um Lernrate, bei zwei Neuronen gleicher Aktivität (–1 bzw. 1)
      - Abschwächung um Lernrate, sonst
    - Lernrate meist 1/n (n = #Neuronen)

Hopfield Netz

- Vorgang der Mustererkennung:
  - Initialisierung durch Eingabevektor(binär)
  - Nachfolgezustand: für jedes Neuron:
    - Aufsummieren aller eingehenden Gewichte: Addition bei aktivem gegenüberliegendem Neuron (Wert 1), Subtraktion bei passivem Neuron (Wert -1)
    - Wenn Summe >= 0 ist Neuron aktiv, sonst passiv
    - Iteriere bis keine Veränderung mehr auftritt

Kohonen SOM

- Unüberwachtes Lernen
- Eingabeschicht mit allen Neuronen der Kohonenschicht verbunden



Kohonen SOM

- Ermittlung des sog. Gewinnerneurons:
  - euklidischer Abstand zw. Eingabevektor und Gewichtsvektor d. Neuronen

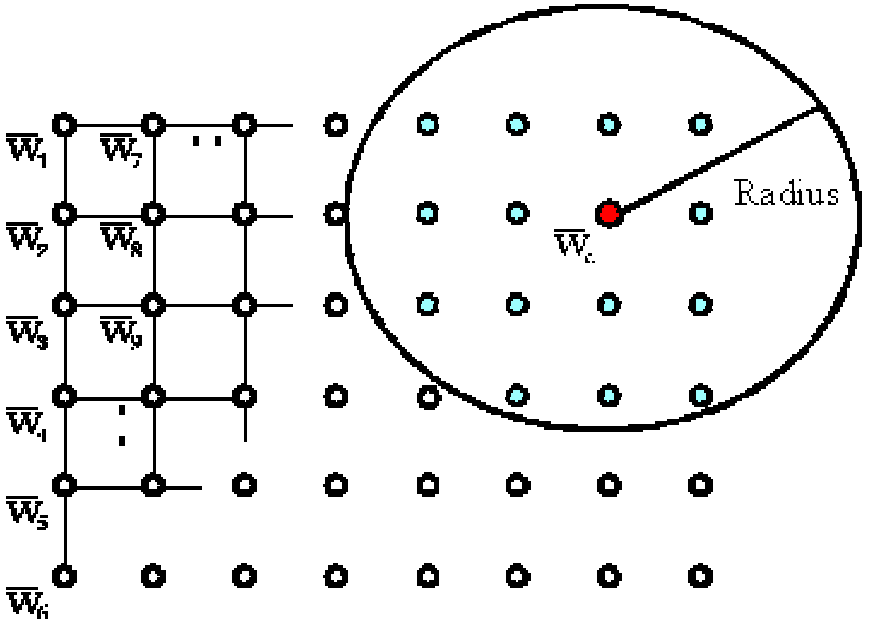
$$\sqrt{(I - W_i)^2}$$

    - Neuron mit minimalem Abstand => Gewinner
- in „Nachbarschaft“ des Gewinners:
  - Anpassung der Gewichtsvektoren

$$\Delta W_i = \alpha(I - W_i)$$

Kohonen SOM

- Nachbarschaftsfunktionen:
  - z.B. Gaußsche Glockenfunktion, Mexikanerhut-funktion, Zylinder und Doppelzylinder



Kohonen SOM

- Größe der Nachbarschaft, sowie Lernrate sinken im Laufe des Trainings
  - Erst Umordnung, dann Konvergenz
- Abbruch meist nach vorgegebener Anzahl an Durchläufen oder wenn Gewichtsänderung

$$\left|\Delta W < \varepsilon\right|$$

Noch Fragen ?

Vielen Dank für Ihre Aufmerksamkeit!