



Mustererkennung in der Bioinformatik

Dr. Hans A. Kestler, Dipl.-Inf. André Müller

WS 2003/2004

Übungsblatt 8

1. Aufgabe (20): Implementierung und Simulation von RBF Netzen

In dieser Aufgabe soll ein RBF-Netz in Matlab implementiert und zur Klassifikation verwendet werden. Gegeben sei eine Trainingsmenge

$$\mathcal{T} = \{(\mathbf{u}^\mu, T^\mu) \mid \mathbf{u}^\mu \in \mathbb{R}^d, T^\mu \in \mathbb{R}^n, \mu = 1 \dots M\}$$

Die Zielvektoren T^μ sind hierbei Vektoren aus dem \mathbb{R}^n . Für Klassifikationsaufgaben müssen die Klassenlabels $\omega^\mu \in \{1 \dots n\}$ des μ -ten Samples folgendermaßen codiert werden:

$$T_i^\mu = \begin{cases} 1 & \text{falls } i = \omega^\mu \\ 0 & \text{sonst} \end{cases}$$

Ein zweischichtiges RBF-Netzwerk sei folgendermassen definiert: Die verdeckte Schicht bestehe aus m RBF-Prototypen mit Zentren $\mathbf{c}_1 \dots \mathbf{c}_m \in \mathbb{R}^d$ und Streuweiten $\sigma_1 \dots \sigma_m \in \mathbb{R}^+$. Das j -te verdeckte Neuron berechnet bei Eingabe von $\mathbf{u}^\mu \in \mathbb{R}^d$ den Ausgangswert

$$h_j^\mu = \exp\left(-\frac{\|\mathbf{u}^\mu - \mathbf{c}_j\|^2}{\sigma_j^2}\right)$$

Die Ausgangsschicht bestehe aus n linearen Neuronen mit Gewichten w_{kj} und Schwellwerten w_{k0} . Die Ausgänge berechnen sich dann zu

$$y_k^\mu = \sum_{j=1}^m h_j^\mu w_{kj} - w_{k0} \quad k = 1 \dots n$$

Zur Bewertung der Netzwerkausgaben, soll die Summe der quadratischen Abweichungen zwischen Netzausgang und Lehrersignal verwendet werden. Diese Zielfunktion ist gegeben durch

$$E(C) = \sum_{\mu=1}^M \sum_{k=1}^n (T_k^\mu - y_k^\mu)^2 \quad .$$

Leiten Sie die Lernregeln für die Gewichte c_{ij}, σ_j und w_{jk} her (*backpropagation learning*). Implementieren Sie das oben beschriebene RBF-Netz in Matlab.

- Als Initialwerte der RBF-Zentren \mathbf{c}_j soll eine zufällige Untermenge der Trainingsdaten verwendet werden.
- Initialisieren sie die Ausgangsgewichte (und den Schwellwert) w_{jk} mit kleinen Zufallswerten aus dem Intervall $(-\delta, \delta)$ ($\delta < 1$). Die initialen Zentrenweiten σ_j sollen auf den λ -fachen ($0 < \lambda < 1$) Euklidischen Abstand zum nächsten Prototypen gesetzt werden.
- Die Optimierung der Gewichte soll mittels Gradientenabstieg erfolgen. Die Lernrate soll linear zwischen den zwei Werten η_0 und η_1 ($\eta_1 < \eta_0$) über die t Trainingsschritte variiert werden, so dass die Lernrate im Trainingsschritt τ ($\tau = 0 \dots t - 1$) $\eta_\tau = \eta_0 + (\eta_1 - \eta_0)\tau/(t - 1)$ beträgt. In jedem Lernschritt wird zufällig ein Datenpunkt aus der Trainingsmenge ausgewählt und nach Berechnung des Gradienten der Fehlerfunktion die Gewichte mit der Lernrate η_τ adaptiert.

- d) Testen Sie Ihre Implementierung mit dem `twospirals` Datensatz (Trainingsdatensatz `twospirals.train` und Testdatensatz `twospirals.test`) und dokumentieren Sie die Ergebnisse (Erkennungsrate auf Trainings- und Testdatensatz). `twospirals` ist ein Zweiklassenproblem. Verwenden Sie ein Ausgangsneuron für jede Klasse und bestimmen Sie mit der *winner takes all* Strategie die Ausgangsklasse des RBF Klassifikators d.h. das neuronale Netz entscheidet sich für die Klasse

$$H(\mathbf{u}) = \arg \max_k y_k(\mathbf{u})$$

mit dem größten Ausgangssignal.

Die `twospirals` Datensätze sind im ASCII-Format. Die erste Zeile enthält eine ganze Zahl, die die Dimension der Eingabevektoren repräsentiert. Jede Zeile enthält einen Eingabevektor und einen Zielvektor (z.B. numerischer Klassenlabel). Jedes Vektorelement ist mit Leerzeichen getrennt.

- e) Stellen Sie den Verlauf der Fehlerfunktion über die Trainingsschritte graphisch dar. Stellen Sie den Verlauf der Klassifikationsrate desselben Laufs graphisch dar.
- f) Testen Sie Ihren Klassifikator auch mit dem reduzierten Golub-Datensatz `golub50.train` bzw. `golub50.test` (diese sind dann wieder im üblichen Format d.h. die letzte Zeile enthält die Klassenlabels).