Neurale Netze und Biostatistik

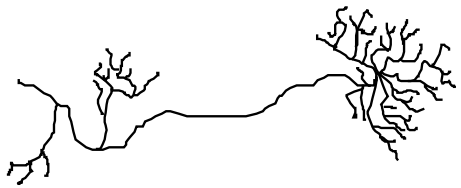
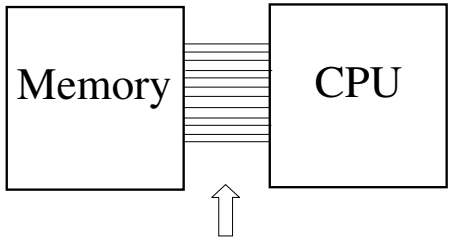
Kurt Hornik

Inhalt

- Einleitung
- Neurale Netze
- Maschinelles Lernen
- Benchmarking

Einleitung

Was sind neurale Netze?

BRAIN	COMPUTER
	
~ 10^{11} Neurons	32 Connections
~ 10^{14} - 10^{15} Connections	~ 10^{11} Transistors
parallel distributed	serial local

„Connectionism“, „Parallel Distributed Processing“

Was sind neurale Netze?

Informationsverarbeitende Systeme die

- Berechnungen auf Basis der Kombination („Vernetzung“) einfacher Berechnungselemente vornehmen;

- sich auf ihre „Umwelt“ einstellen können (und in diesem Sinne „lernfähig“ sind)

Berechnungselemente e.g. McCulloch-Pitts Neuronen $x \mapsto H(a'x - \theta)$.

Arten von Lernen

Supervised Learning Gewünschter Output („Target“) für gegebenen Input bekannt, erlernt wird zugrundeliegende Funktion (Regression, Klassifikation)

Unsupervised Learning Keine Targets, das Netz versucht Struktur in den Inputdaten zu finden (Clusteranalyse, . . .)

Reinforcement Learning Keine expliziten Targets, aber Information über richtig/falsch oder besser/schlechter

Das ABC für neurale Netze

- **B**iological neural nets
- **A**rtificial neural nets (qualitativ/symbolisch, biologische Relevanz?)
- **C**omputational neural nets (quantitativ/numerisch)

Artificial Intelligence versus Computational Intelligence

Geschichte

1943 McCulloch (Psychiater und Neuroanatom) and Pitts (Mathematiker) präsentieren logischen Kalkül für neurale Netze

1949 Hebb's Buch *The Organization of Behavior*: erste explizite Lernregel

1958 Rosenblatt erfindet das *Perceptron*: neuer Zugang zur Mustererkennung, Perceptron Convergence Theorem

1969 Minsky and Papert zeigen Grenzen des Perzeptrons auf

1986 (Wieder)Entdeckung des *Backpropagation Algorithmus* durch Rumelhart, Hinton, Williams

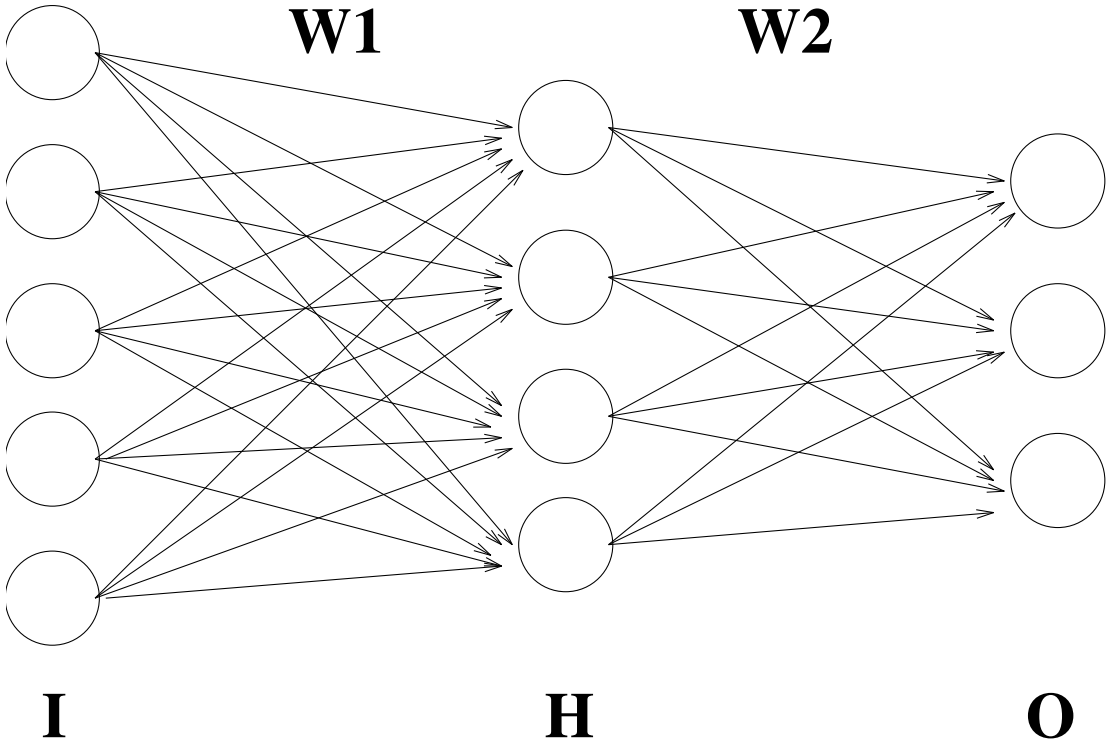
Was ist maschinelles Lernen?

- Teilgebiet der Informationswissenschaften das sich mit „automatischem“ Lernen befasst

- Das was die Machine Learning Community treibt . . .

Neurale Netze

Mehrschicht-Perzeptrone (MLPs)



Dreischicht-Perzeptrone

Auch „Single Hidden Layer Perceptrons“ (SHLPs), oder einfach „neural networks“

Implementieren Funktionen der Form

$$x \mapsto \sum_{i=1}^{k} \beta_i \psi(a_i' x + \theta_i)$$

Approximationseigenschaften

(I.e., wie „mächtig“ ist das Berechnungsmodell?)

Hornik (1993): Falls Aktivierungsfunktion ψ in einem offenen Intervall nicht polynomial ist, ist die Menge der obigen Funktionen mit beliebigem k und Parametern a, θ und β dicht in $L^p(\mu)$ für μ mit kompaktem Träger, $1 \leq p \leq \infty$.

(Varianten für Approximation von stetigen Funktionen etc.)

Training

Üblicherweise durch (näherungsweise) Minimierung des MSE

$$E_n(w) = \frac{1}{n} \sum_{i=1}^{n} \|y_i - f(x_i, w)\|^2$$

(wobei w der Parametervektor aller Netzwerkgewichte a, θ und β ist).

Im Klassifikationsfall üblicherweise „1-aus-N Kodierung“

Traningsalgorithmen

Bei Verwendung von on-line gradient descent $\Rightarrow$ Error Back Propagation, gilt allgemein für „feedforward“ Netze: rekursiv Berechnung vom Input zum Output, Anpassung vom Output zum Input.

Einfaches Gradientenverfahren (biologisch plausibel?) $\Rightarrow$ Vielzahl von Erweiterungen/Modifikationen (Momentum Term, Bold Driver Technique, Delta-Delta Regel, Delta-Bar-Delta, Quickprop, Conjugate Gradients, Newton-Typ, . . .)

Overfitting et al

Wie „groß“ soll das verwendete SHLP sein? Problem des möglichen Auswendiglernens der Stichprobe anstatt des Erlernens funktionaler Zusammenhänge in der Grundgesamtheit.

Allgemein: Mimimierung des unbekannten „wahren“ Risikos $R(f)$ auf Basis des bekannten empirischen Risikos $R_n(f)$, wobei e.g.

$$R(f) = E(Y - f(X))^2, \qquad R_n(f) = \frac{1}{n}\sum_{i=1}^{n}(y_i - f(x_i))^2$$

für geeignete Klassen von Modellfunktionen f.

Overfitting et al (Fortsetzung)

Im wesentlichen 3 Möglichkeiten:

- Explizite Bestrafung von zu hoher Komplexität durch Regularisierungsterm in $R_n(f)$ (aber nicht „Größe“)
- Aufteilen der Daten in Trainings- und Testset (oder Training, Test und Validation Set; bei „großen“ Datensätzen)
- Kreuzvalidierung

Software

Funktion nnet() in Standard-Package nnet

Beispiel aus ?nnet . . .

Learning Vector Quantization

Partitionierung durch Lernen von Prototypen sodass $\sum_i d(x_i, c(x_i))$ minimal wird. Bei einfacher LVQ: zufällige Initialisierung von k Prototypen, dann wiederhole:

1. wähle zufällig Datenpunkt x_i und den nächsten Prototyp („Winner“) $c(x_i)$.

2. verschiebe den Prototyp in Richtung Datenpunkt:

$$\Delta c(x_i) \propto x_i - c(x_i)$$

Erweiterungen

- Auch für Klassifikation geeignet: der Winner muß dann der nächste Prototype aus der richtigen Klasse sein

- LVQ Basisalgorithmus auch „hard competitive learning“ oder „winner-takes-all“

- Falls mehrere Prototypen verschoben: „soft competitive learning“ oder „winner-takes-most“

Software

Funktionen lvq1() et al. in Package class

Self-Organizing Maps

On-line Version: initialisiere „Gitter“ $\{w_j\}$ in geeignetem topologischen Raum; wiederhole

1. wähle zufällig Datenpunkt x_i und den nächsten Prototyp („Winner“) $c(x_i)$.

2. verschiebe Winner und alle Prototypen in geeigneter Umgebung in Richtung x_i.

(Form von mehrdimensionaler Skalierung).

Maschinelles Lernen

Support Vector Maschinen

Kontrolle von Overfitting via „Komplexität“, gemessen e.g. durch maximalen Abstand zwischen wahrem und empirischen Risiko

Komplexitätsmaß *VC-Dimension* (maximales m sodass es eine Stichproben der Größe m gibt die auf alle 2^m möglichen Arten zerlegt werden kann).

Mit Wahrscheinlichkeit $\geq 1 - \eta$:

$$R(w) \leq R_n(w) + \Phi(\text{VCdim}, n, \eta)$$

Optimale Hyperebenen

Betrachten binäre Klassifikationsaufgabe mit $y_i \in \{-1, 1\}$ und linearen Klassifikatoren $f(x, w) = \text{sgn}(a'x - \theta)$.

Falls linear separabel, gibt es w mit $y_i f(x_i, w) > 0$ für alle i. Optimal? Möglichst großer Abstand der Daten von der Hyperebene, aber bei kontrollierter VC-Dimension. Motiviert:

$$\|a\|^2 \to \min, \qquad y_i f(x_i, w) \geq 1, \quad i = 1, \ldots, n.$$

Optimale Hyperebene durch quadratische Programmierung, löst lineares Gleichungssystem mit inneren Produkten der x_i als Koeffizienten.

Allgemeiner Fall

Falls nicht exakt trennbar

- Einführung von Schlupfvariablen und/oder
- Transformation in geeignet hochdimensionalen Raum (e.g., Polynome) in dem linear trennbar

Verwendung des „Kernel Trick“ zur Berechnung der entsprechenden inneren Produkte ($\langle T(x), T(y) \rangle = K(x, y)$).

Software

Funktion svm() von D. Meyer als Interface zu libsvm in Package e1071; Package svlab von A. Karatzoglou et al. (in statu nascendi).

Kombinationsstrategien: Motivation

Flexible statistische Verfahren (Verwendung flexibler Klassen von Modellfunktionen: CART, NN/MLP, . . .) typischerweise:

- „instabil“ (gering veränderte Daten geben stark unterschiedliche Lösungen)
- „zufällig“ (zufällige Startwerte)
- Probleme mit lokalen Minima

Attraktiv, aber wie zuverlässig sind Lösungen?

Idee: Daten und/oder Startwerte verändern und erhaltenen Lösungen „kombinieren“

Bagging

Bootstrap Aggregating (Breiman)

Gut wären viele unabhängige Stichproben (aus zugrundeliegender Verteilung). Ersatzweise: viele unabhängige Stichproben aus empirischer Verteilung (Bootstrap Stichproben)

Algorithmus:

- Generiere $\mathcal{Z}_1, \ldots, \mathcal{Z}_M$ durch Ziehen mit Zurücklegen aus $\mathcal{Z}$
- Schätze $h_1, \ldots, h_M$
- geeignete Kombination, e.g. durch „Mehrheitsentscheidung“ (simple majority vote) bei Klassifikationsaufgaben

Boosting

AdaBoost (Freund & Schapire): binäre Klassifikationsaufgabe mit $Y = \{-1, 1\}$. Initialisiere $w_i = 1/n$ und wiederhole

1. Schätze Klassifikationsfunktionen $h_m : X \to \{-1, 1\}$ unter Verwendung der Gewichte w auf der Originalstichprobe $\mathcal{Z}$ („reweighted“) oder einer gemäß w aus $\mathcal{Z}$ gezogenen neuen Stichprobe („resampled“)

2. $\epsilon_m = E_w I(h_m(x) \neq y)$, $\beta_m = \log((1 - \epsilon_m)/\epsilon_m)$

3. $w_i \leftarrow w_i \exp(\beta_m I(h_m(x_i) \neq y_i))/S$

Boosting (Fortsetzung)

Ergebnis ist „weighted majority vote“ $\text{sgn}(\sum_m \beta_m h_m)$.

Interpretierbar als stagewise functional gradient descent zur Minimierung von $E(e^{-yH(x)})$; Lösung

$$H(x) = \frac{1}{2} \log \frac{P(y=1|x)}{P(y=-1|x)}.$$

Warum $E(e^{-yH(x)})$? Differenzierbare obere Schranke für Misklassifikationsrate.

Verallgemeinerung . . . . . .

Software

Funktion bagging() in Package ipred; Package gbm

Benchmarking

Motivation

Q: Welches Verfahren soll man jetzt verwenden?

A: Das welches am besten funktioniert.

Notwendigkeit des systematischen Benchmarking!

Standarddatensätze aus dem UCI Machine Learning Repository

`http://www.ics.uci.edu/~mlearn/MLRepository.html`

Probleme

- Hyperparameter tuning versus off-the-shelf
- Messung und Vergleich der erhaltenen Ergebnisse (e.g., Mittelwert versus Median der erhaltenen Performancemaße)
- Verallgemeinerung?

Design

For jeden (Real)Datensatz, 100 Training und Test Sets via 10 Wiederholungen von 10-facher Kreuzvalidierung (i.e., 10 Partitionen mit disjunkten Test Sets von 10 Permutationen der Daten).

Hyperparameter Tuning auf Basis von 1/3 des Training Set als Validation Set.

Daten

BreastCancer Vorhersage von benign/malign auf Basis zellulärer Charakteristika (699 Beobachtungen, 10 Prädiktoren)

Heart (UCSD Medical Center) Bestimmung von Herzinfarkpatienten mit hohem Risiko

PimaIndiansDiabetes Pima Indianer; Vorkommen von Diabetes auf Basis von medizinischen Daten, Alter und Schwangerschaft (768 Beoachtungen, 8 Prädiktoren)

Ionosphere Johns Hopkins University Ionosphere Daten: Vorkommen freier Elektronen in der Ionosphäre. 2 Klassen, 351 Beobachtungen, 35 Prädiktoren)

Ergebnisse

	B.C.	Heart	PI.D.	Iono
nnet	4.49	14.50	23.73	12.13
lvq	4.89	20.62	28.33	14.38
svm	3.14	15.87	23.53	5.93
bagg	3.21	14.93	24.09	8.32
rForst	2.28	14.15	23.60	7.32
lda	3.56	13.67	22.60	12.99
glm	5.32	14.66	22.37	11.91
rpart	5.51	18.52	25.38	12.80