Non-parametric Learning of Weights

- Perceptron (linear) and LVQ (non linear)
- Relaxation Procedures
- Widrow – Hoff (LMS, Adaline)
- Ho-Kashyap Procedure
- Incremental Gauss – Newton = RLS
- Fisher's linear discriminant
- Support Vector Machines

One can trace the history of Neural networks to the work of Santiago Ramon y Cajal, who discovered that the basic building element of the brain is the neuron. The brain comprises approximately 60-100 billion neurons! Each neuron is connected with other neurons via elementary structural and functional units/links, known as synapses. It is estimated that there are 50-100 trillion synapses. These links mediate information between connected neurons.

The most common type of synapses are the chemical ones, which convert electrical pulses, produced by a neuron, to a chemical signal and then back to an electrical one. Depending on the input pulse(s), a synapse is either activated or inhibited. Via these links, each neuron is connected to other neurons and this happens in a hierarchical way, in a layer-wise fashion.

In 1943, Warren McCulloch and Walter Pitts, developed a computational model for the basic neuron linking neurophysiology with mathematical logic. They showed that given a sufficient number of neurons and adjusting appropriately the synaptic links, each one represented by a weight, one can compute, in principle, any computable function. As a matter of fact, it is generally accepted that this is the paper that gave birth to the fields of neural networks and artificial intelligence.

Frank Rosenblatt borrowed the idea of a neuron model, as suggested by McCulloch and Pitts, and proposed a true learning machine, which learns from a set of training data. In its most basic version, he used a single neuron and adopted a rule that can learn to separate data, which belong to two linearly separable classes. That is, he built a Pattern Recognition system. He called the basic neuron a perceptron and developed a rule/algorithm, the perceptron algorithm, which we used in HW 1 and review it briefly.

Discuss a neuron and a relay … idealization of $\sigma(c,\underline{x}) = \dfrac{1}{1+\exp(-c\underline{w}^T\underline{x})} \ldots as\ c \to \infty, step = g(\underline{x})$

$$\frac{d\sigma(c,x)}{dx} = c\sigma(c,x)(1-\sigma(c,x))$$

$$when\ use\ mse\ criterion\ or\ cross-entropy$$

$$gradient = -\sum_{n=1}^{N} e_n \underline{x}_n = -\sum_{n=1}^{N} (z_n - \sigma(c,\underline{x}_n))\underline{x}_n \sigma(c,\underline{x}_n)(1-\sigma(c,\underline{x}_n))$$

Formalization:

Select class 1 if $\underline{w}^T \underline{x} > 0;\ \underline{x} = (-1\ x_1\ x_2 \cdots\cdots x_p)$

Look at it geometrically by recalling that the inner product is related to cosine of the angle between vectors.

$$\underline{w}^T \underline{x} > 0 \text{ if } \underline{x} \text{ belongs to class 1} \qquad \Rightarrow \theta < 90^\circ \text{ (acute angle)}$$
$$\underline{w}^T \underline{x} < 0 \text{ if } \underline{x} \text{ belongs to class 2} \qquad \Rightarrow \theta > 90^\circ \text{ (obtuse angle)}$$
$$\text{since } \underline{w}^T \underline{x} = \|w\| \|\underline{x}\| \cos\theta$$

You can setup an optimization problem to minimize the number of errors or a measure of errors (e.g., how far away from 0).

$$\sum_{n=1}^{N} [z_n \max(0, b_n - \underline{w}^T \underline{x}_n) + (1 - z_n) \max(0, \underline{w}^T \underline{x}_n + b_n)]$$
$$b_n = 0 \Rightarrow \sum_{n=1}^{N} [z_n \max(0, -\underline{w}^T \underline{x}_n) + (1 - z_n) \max(0, \underline{w}^T \underline{x}_n)]$$

Error metric: $\Rightarrow z_n = 1 \,\&\, \underline{w}^T \underline{x}_n > 0 \Rightarrow no\ cost\ \&\ z_n = 1 \,\&\, \underline{w}^T \underline{x}_n < 0 \Rightarrow gradient\ -\underline{x}_n$

$$z_n = 0 \,\&\, \underline{w}^T \underline{x}_n < 0 \Rightarrow no\ cost\ \&\ z_n = 0 \,\&\, \underline{w}^T \underline{x}_n > 0 \Rightarrow gradient\ \underline{x}_n$$
$$same\ conclusions\ apply\ with\ nonzero\ b_n!$$

Perceptron: Supervisory Learning (Reinforcement Learning) using **incremental gradient** or **stochastic gradient**… Update <u>w</u> only if you make mistakes

$$
\begin{array}{l}
Do\ until\ errors\ stabilize \\
\quad Do\ n = 1 : N \\
\quad\quad if\ e_n \neq 0 \\
\quad\quad\quad \underline{w} \leftarrow \underline{w} + \eta e_n \underline{x}_n \\
\quad\quad end\ if \\
\quad end \\
end
\end{array}
$$

$$if\ z_n = 1 \text{ and } g(\underline{x}_n) = 0, e_n = 1 \Rightarrow \underline{w}^T \underline{x}_n \leftarrow \underline{w}^T \underline{x}_n + \eta \underline{x}_n^T \underline{x}_n \uparrow \text{ as it should!}$$
$$if\ z_n = 0 \text{ and } g(\underline{x}_n) = 1, e_n = -1 \Rightarrow \underline{w}^T \underline{x}_n \leftarrow \underline{w}^T \underline{x}_n - \eta \underline{x}_n^T \underline{x}_n \downarrow \text{ as it should!}$$

Perceptron learning rule (Incremental gradient, Stochastic gradient)

Alternate: Compute mini-batch gradient

$$
\begin{aligned}
&Do\ until\ errors\ stabilize\\
&\quad Do\ n = 1 : N\ in\ \ steps\ of\ \ m\\
&\qquad \underline{w} \leftarrow \underline{w} + \eta \sum_{j=n}^{n+m-1} e_n \underline{x}_n\\
&\qquad end\ if\\
&\quad end\\
&end
\end{aligned}
$$

Perceptron Convergence Theorem:

For simplicity, scale class 0 samples as $\underline{x}^{\,i} = -\underline{x}^{\,i}$

Go through the two proofs

Optimal Learning rates or relaxation method

Extension to Multiple Classes

Key properties of Perceptron:

- It employs distributed decision-based credit assignment
- Update weights only when misclassification occurs
- Distributed and localized: reinforce correct class, penalize wrong decision class
- Update depends on incremental/stochastic gradient

How many random patterns a Perceptron with p inputs can learn reliably in a 2 class case? *2p*

$P(N,p) = \begin{cases} 1 & N \le p+1 \\ \dfrac{2}{2^N} \displaystyle\sum_{i=0}^{p} \binom{N-1}{i} & N > p+1 \\ \approx \Phi\left(\dfrac{2p-N}{\sqrt{N}}\right) & \text{for large } N \end{cases}$	$N = 4; p = 2$ $\Rightarrow P(N,p) = \frac{1}{8}(1+3+3)$ $= \frac{7}{8}$ *XOR* Pattern cannot be correctly classified by a Perceptron

Back Propagation:

$$L(x) = L(f(h(g(x)))$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial f}\frac{\partial f}{\partial h}\frac{\partial h}{\partial g}\frac{\partial g}{\partial x} = \lambda_g \frac{\partial g}{\partial x}$$

$$\lambda_f = \frac{\partial L}{\partial f}; \lambda_h = \frac{\partial L}{\partial h} = \lambda_f \frac{\partial f}{\partial h};$$

$$\lambda_g = \frac{\partial L}{\partial g} = \lambda_h \frac{\partial h}{\partial g}$$

$$x \to g \to h \to f \to L$$

$$\frac{\partial L}{\partial x} = \lambda_g \frac{\partial g}{\partial x} \leftarrow \lambda_g = \frac{\partial L}{\partial g} = \lambda_h \frac{\partial h}{\partial g} \leftarrow \lambda_h = \frac{\partial L}{\partial h} = \lambda_f \frac{\partial f}{\partial h} \leftarrow \lambda_f = \frac{\partial L}{\partial f}$$

Decision Trees:

- Developed since 1960s and popularized by Brieman and Quinlan
  - ⇨ Tree Algorithms: ID3, C4.5, C5.0 and CART
- Three uses: Data description (compression, rules), classification and generalization
- Popular in statistics, pattern recognition, decision theory, signal processing and machine learning
- Have been used in industrial applications, particularly in diagnosis and quality control. For example, look at my papers on sequential fault diagnosis since 1990. Primarily in IEEE T-SMC.
- Why? Invariant to scaling; Can handle large datasets; Easily ignore redundant variables; Handle missing variables through surrogate splits; Easy to interpret and explain
- Nearly half of the data mining competitions are won by using some variants of tree ensemble methods: Boosting, Random Forests, Bagging
- Surveys: Srirama K. Murthy: "Automatic Construction of Decision Trees From Data: A Multi-disciplinary Survey," pp 1-49. Kulwer Academic Publishers. Data Mining and Knowledge Disocery 2 (4): 345-389 (1998)

  S. B. Kotsiantis, "Decision trees: a Recent Overview," Artificial Intelligence Review, 39:261–283, 2013.

Tests: single attribute tests; hyperplane tests

Continuous tests: Ranges

Select t such that mutual information IG(z|x,t) is maximum

$$IG(z\,|\,x,t) = H(z) - H(z\,|\,x,t)$$
$$H(z) = -\sum_{j=1}^{C} P(z=j)\log_2 P(z=j)$$
$$H(z\,|\,x,t) = P(x<t)H(z\,|\,x<t) + P(x\ge t)H(z\,|\,x\ge t)$$

$z \in \{1,2,..,C\};\ \underline{x} \in R^p;\ p(\underline{x}) = \sum_{i=1}^{C} P[\underline{x}, z=i] = \sum_{i=1}^{C} P(z=i)\,p(\underline{x}\,|\,z=i) = \sum_{i=1}^{C} \pi_i N(x;\mu_i,\sigma_i^2)$

Recall for a binary test on x with a theshold $t: IG(z\,|\,x,t) = H(z) - H(z\,|\,x,t)$

$$H(z) = -\sum_{j=1}^{C} P(z=j)\log_2 P(z=j) = -\sum_{j=1}^{C}\pi_j \log_2 \pi_j$$

$$H(z\,|\,x,t) = P(x<t)H(z\,|\,x<t) + P(x\ge t)H(z\,|\,x\ge t) = \left(\sum_{i=1}^{C}\pi_i\Phi(\frac{t-\mu_i}{\sigma_i})\right)H(z\,|\,x<t) + \left(\sum_{i=1}^{C}\pi_i(1-\Phi(\frac{t-\mu_i}{\sigma_i}))\right)H(z\,|\,x\ge t)$$

where $\Phi(c) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{c} e^{-u^2/2}du = CDF$ of a standard Normal (Gaussian) distribution.

Note that by Bayes rule, $P(z=j\,|\,x<t) = \dfrac{P(x<t\,|\,z=j)P(z=j)}{P(x<t)} = \dfrac{\Phi(\frac{t-\mu_j}{\sigma_j})\pi_j}{\sum_{i=1}^{C}\pi_i\Phi(\frac{t-\mu_i}{\sigma_i})}$

So, $H(z\,|\,x<t) = -\sum_{j=1}^{C}\left(\dfrac{\Phi(\frac{t-\mu_j}{\sigma_j})\pi_j}{\sum_{i=1}^{C}\pi_i\Phi(\frac{t-\mu_i}{\sigma_i})}\right)\log_2\left(\dfrac{\Phi(\frac{t-\mu_j}{\sigma_j})\pi_j}{\sum_{i=1}^{C}\pi_i\Phi(\frac{t-\mu_i}{\sigma_i})}\right); H(z\,|\,x\ge t) = -\sum_{j=1}^{C}\left(\dfrac{[1-\Phi(\frac{t-\mu_j}{\sigma_j})]\pi_j}{\sum_{i=1}^{C}\pi_i[1-\Phi(\frac{t-\mu_i}{\sigma_i})]}\right)\log_2\left(\dfrac{[1-\Phi(\frac{t-\mu_j}{\sigma_j})]\pi_j}{\sum_{i=1}^{C}\pi_i[1-\Phi(\frac{t-\mu_i}{\sigma_i})]}\right)$

So, $IG(z\,|\,x,t) = -\sum_{j=1}^{C}[\pi_j\log_2\pi_j - \Phi(\frac{t-\mu_j}{\sigma_j})\pi_j\log_2\left(\dfrac{\Phi(\frac{t-\mu_j}{\sigma_j})\pi_j}{\sum_{i=1}^{C}\pi_i\Phi(\frac{t-\mu_i}{\sigma_i})}\right) - [1-\Phi(\frac{t-\mu_j}{\sigma_j})]\pi_j\log_2\left(\dfrac{[1-\Phi(\frac{t-\mu_j}{\sigma_j})]\pi_j}{\sum_{i=1}^{C}\pi_i[1-\Phi(\frac{t-\mu_i}{\sigma_i})]}\right)]$

Discuss Entropy

Gini Index

Mutual Information

JMI

Decision Trees: Low Bias and High Variance

Cost-complexity pruning

CART

Bagging and Random Forest

Boosting: AdaBoost and Gradient Boosting

Missing Value Problem

ECC

Examples