Simple classifiers

- One relevant attribute
- Error correction learning
- N Nets: Nonlinear classifiers
One relevant attribute

For each attribute,
    For each value of that attribute, make a rule:
        count how often each class appears
        find the most frequent class
        rule assigns this class to this attribute-value.
    Calculate the error rate of the rules.
Choose the rules with the smallest error rate.

- WEKA/classifiers/OneR
- WEKA/classifiers/ZeroR
  - Predicts the mean/mode value
Squared error criterion

\[ E = \sum_{p \in \text{TrainingSet}} E_p \]

\[ E_p = \frac{1}{2} \sum_{j \in \text{Out}} (t_j - x_j)^2 \]
McCulloch-Pitts neuron [1943]

- Neuron-like computing elements
- Networks are Turing equivalent

\[ \phi \left( \sum_j x_j w_{ji} \right) \]

\[ \phi \left( \sum_j x_j w_{ji} \right) = o_j \]
**Perceptron [Rosenblatt 1961]**

- Two layer network
- Convergence procedure
- Perceptron learning Theorem
Two layer network

Input $x$, output $f(x)$

$f(x)$

$X$
### Convergence procedure

- Assume training set \(((x^p, t^p = f(x^p))\)
- Delta learning rule

\[
\delta_j^p \equiv t_j^p - x_j^p
\]

\[
\Delta w_{ji}^p = \eta \ x_j^p \ \delta_j^p
\]
Linear separator

- [Nilsson96]
- hyperplane separating input space $x$
- Normal to hyperplane

\[ x \cdot W + w_{n+1} = 0 \]
\[ x \cdot n + \frac{w_{n+1}}{|W|} = 0 \]

\[ n = \frac{W}{|W|} \]

Unit vector normal to hyperplane

Figure 4.2: TLU Geometry
Error-correcting moves

- solution space is hyperwedge, extending from origin
- Experience sequence $y_1=+, y_2=-, y_3=+, y_4=+$
- Each move is wrt that hyperplane’s normal
- “Polarity” of hyperplane reflects desired output

Figure 4.7: Moving Into the Solution Region
## Widrow-Hoff [1960]

- aka Delta, LeastMeansSquare (LMS)
- Quadratic error measure ensures can minimize in direction of steepest descent (vs. normal to existing hyperplane)
- Incremental (wrt/ single training instance) vs. batch updating
NNets: Nonlinear classifiers

- Linear separability [Minsky&Papert 1963]
- Back propagation
- Learning in NNets with state
- WEKA/classifiers/neural/*
Linear separability
[Minsky&Papert 1963]

- Two layer perceptron only capable of realizing linearly separable classifications
- Important functions (e.g., XOR, parity) solutions not realizable
- Since no learning procedure exists for them, multi-layer perceptrons are “barren” extension
Back propagation - Overview

Generalization of Rosenblatt (also Widrow, Huff) learning rule to networks with “hidden layers,” allowed by differentiable squashing function

\[ \partial(x) = t(x) - f(x) \]

Diagram:

- \( x \)
- Activity flow
- Error flow
Chain rule

- Use chain rule to describe change in error as a function of weight change as the product of change as a function of net input, and change in net input as a function of the particular weight's:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial o_j} \frac{\partial o_j}{\partial w_{ji}}$$
The second term is easy

- Since it depends only on input's activity:

\[
\frac{\partial o_j}{\partial w_{ji}} = \sum_k w_{jk} x_k = x_i
\]
First term

- Post-squash change must itself be described as product of change in error as function of unit's output, and change in unit's output as function of changing input:

\[
\frac{\partial E}{\partial o_j} = \frac{\partial E}{\partial x_j} \frac{\partial x_j}{\partial o_j}
\]
Second term

Again easier, being exactly the derivative of the squashing function:

\[ \frac{\partial x_j}{\partial o_j} = \frac{\partial \varphi(o_j)}{\partial o_j} = \varphi'(o_j) \]

Eg, if we use a (typical) logistic function:

\[ \varphi(o) = \frac{1}{1 + e^{-o}} \]

\[ \varphi'(o) = o(1 - o) \]
If unit receives direct teaching feedback:

Change in error as function of unit's output is straight-forward

\[ E = \frac{1}{2} \sum (t_j - x_j)^2 \]

\[ \frac{\partial E}{\partial x_j} = (t_j - x_j) \]

\[ = \delta_j \]
Non-output units

- Recursively apply the same chain-rule derivation to each of the units it activates:

\[
\frac{\partial E}{\partial x_j} = \sum_{k \in \text{OutNbr}} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial x_j}
\]

\[
= \sum_k \frac{\partial E}{\partial o_k} \frac{\partial}{\partial o_j} \sum_l w_{kl} x_l
\]

\[
= \sum_k \frac{\partial E}{\partial o_k} w_{kj}
\]

\[
= \sum_k \delta_k w_{kj}
\]
Putting it all together:

\[ \delta_j = \begin{cases} 
\varphi'(o_j) (t_j - x_j) & \text{if unit}_j \text{ is output} \\
\varphi'(o_j) \sum_k \delta_k w_{kj} & \text{otherwise}
\end{cases} \]
Hence, to implement a gradient descent on $E$:  

$$\frac{\partial E^p}{\partial w_{ji}} = -x_i^p \delta_j^p$$

$$\Delta w_{ji} = \eta x_i^p \delta_j^p$$
Local changes and global convergence

• In practice, such "online" changes to each training exemplar $p$ may not converge even though cognitively less-plausible "batch" training (where all learning changes are accumulated but not made until entire training corpus presented) will.

• Alternatively a (second order) "momentum" term can also be added to the learning rule:

$$\Delta w'_{ji} = \eta x_j^p \delta_i^p + \alpha \Delta w_{ji}$$
BProp network architectures

- Number of hidden units
- Less than complete connectivity
- Architectural compositions
- Recurrent connections
Number of hidden units
Less than complete connectivity
Architectural compositions

Speech

Forward Model

"Plan"

Inverse Model

Intended speech
Recurrent connections

1:1 copy
Learning in NNets with state

• Elman/Jordan NNets
• Recurrent BackProp