LINEAR NETWORKS FOR DATA-DRIVEN CLASSIFICATION

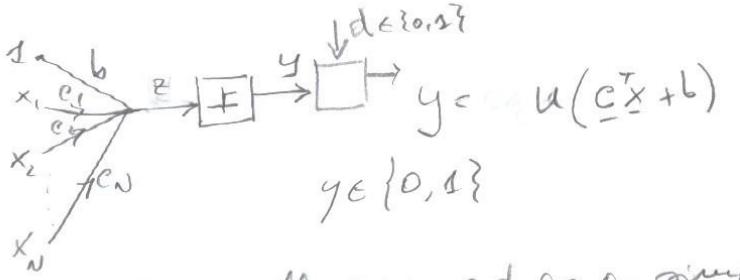
Prof. FRANCESCO A.N. PALMIERI
UNIVERSITÀ DELLA CAMPANIA
"LUIGI VANVITELLI"

[CORSO DI SIGNAL PROCESSING & DATA FUSION oct. 2023]

LINEAR NETWORKS FOR CLASSIFICATION

BINARY CLASSIFIER WITH HARD THRESHOLD

The simplest neural network for classification is the linear perceptron shown in the figure.



The perceptron, originally proposed as a simplified model of a neuron (superposition and thresholding) implements a linear binary classifier. We have found already the structure when we derived model-based classifiers for gaussian or exponential-family generative models.

The network here is ~~ready~~ formulated and must be learned from the training set. More specifically we have a training set of pairs

$$\mathcal{T} = \{(\underline{x}[n], d[n]), n = 1, \dots, n_2\}, \quad d[n] \in \{0,1\}$$

and we seek the parameters $\underline{c}$ and b that minimize the classification error i.e. the events $y[n] \neq d[n]$.

The cost function can be written as

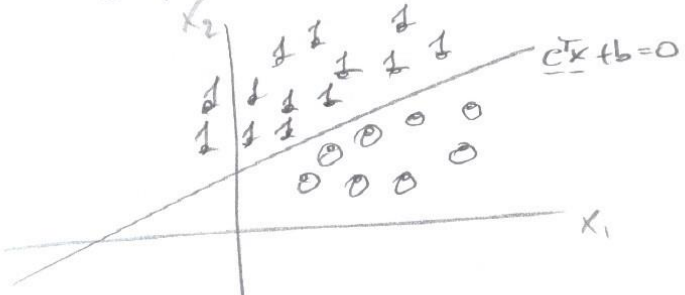
$$\mathcal{E}(\underline{c}, b) = \sum_{n=1}^{n_2} |d[n] - y[n]| = \sum_{n=1}^{n_2} |d[n] - \underline{c}^T \underline{x}[n] - b|$$

$\mathcal{E}(\underline{c}, b)$ counts the number of errors.

Note that since $y[n] - d[n] \in \{-1, 0, 1\}$ the cost can be written also as a sum of squares

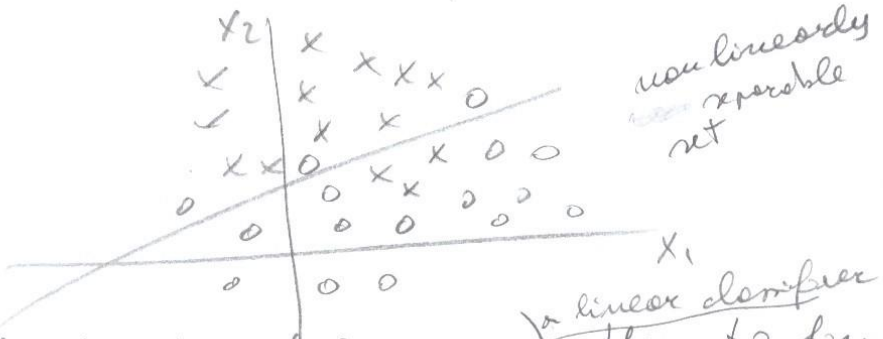
$$E(\underline{c}, b) = \sum_{n=1}^{N} \left(d[n] - \underline{c}^T \underline{x}[n] - b\right)^2$$

The classifier can be visualized in the following figure in 2D where the hyperplane



$\underline{c}^T\underline{x}+b=0$ divides the space in two regions. The points above produce output $y=1$ and the ones below, $y=0$.

Clearly the examples in the training set (or in the validation/test set) may not be such that there exist an hyperplane that separates exactly the two classes (linearly separable)



The objective of learning is then to find a linear classifier the best parameters $(\underline{c}, b)$ that leave the smallest number of examples on the wrong side of the hyperplane.

Unfortunately, there is no closed-form solution to the problem (it is in linear regression), but we can invoke iterative procedures that adjust progressively the position of the hyperplane until the best configuration is found. (LE3)

For easier notation we use, from now on, the augmented vectors $\underline{x}_a = \begin{bmatrix} 1 \\ \underline{x} \end{bmatrix}$ and $\underline{w} = \begin{bmatrix} b \\ \underline{w} \end{bmatrix}$ so that

$$y[n] = u(\underline{w}^T \underline{x}_a[n])$$

($u(\xi)$ is the indicator function = 1 for $\xi > 0$ and 0 for $\xi \leq 0$)

The perceptron rule is a simple iterative algorithm that adjusts $\underline{w}$ progressively to the best linear classifier (Rosenblatt, 1962). The steps are the following

1. Initialize $\underline{w}$ with random values
2. Present the examples $(\underline{x}_a[n], d[n])$ of the training set one at a time, i.e. for each example
   - if $y[n] = d[n]$, $\underline{w}[n] = \underline{w}[n-1]$ (no change)
   - else if $y[n] = 1$ and $d[n] = 0$, $\underline{w}[n] = \underline{w}[n-1] - \mu \underline{x}_a[n]$
   - else if $y[n] = 0$ and $d[n] = 1$, $\underline{w}[n] = \underline{w}[n-1] + \mu \underline{x}_a[n]$
3. Once the whole training set has been presented, take the resulting $\underline{w}$ and use it as initial condition for a new epoch (perhaps shuffling the training set order).
4. Stop at convergence of $\underline{w}$ and $\mathcal{E}(\underline{w})$, i.e. when there are no more significant changes.

The algorithm can be compactly written as the update

$$\underline{w}[n] = \underline{w}[n-1] + \mu \underline{x}_a[n] \left(d[n] - u(\underline{w}^T[n-1] \underline{x}_a[n]) \right)$$

because the error $d[n] - u(\underline{w}^T[n-1] \underline{x}_a[n])$ can only be $-1, 0$ or 1.

Note that in the perceptron algorithm we have included a stepsize parameter μ (that) just as in any gradient (LeCun) algorithm, must be positive. It could be adopted $\mu[n]$, but in the original algorithm proposed by Rosenblatt it was fixed to $\mu=1$, even if it is necessary to keep it small with $0<\mu<1$.

The proof of convergence of the algorithm is very interesting and is one of the few rigorous results that can be invoked in the neural network literature. It is obtained (unfortunately) under the assumptions that the examples are linearly separable, that $\mu=1$ and $\underline{w}[0]=\underline{0}$. We direct the interested student to the specific literature [Haykin, 2006] limiting ourselves here to a few key observations that relate this algorithm to a gradient search.

Note that when a pattern $\underline{x}_a[n]$, that should be classified with $d[n]=1$, and gives $y[n]=0$, the update is

$$\underline{w}[n] = \underline{w}[n-1] + \mu\, \underline{x}_a[n]\,[1-0]$$

Now if re-use the same pattern $\underline{x}_a[n]$ with the new $\underline{w}[n]$ (this is called a-posteriori update) we have

$$\underline{w}[n] = \underline{w}[n-1] + \mu\, \underline{x}_a[n]\,\left[1 - u\left(\left(\underline{w}[n-1] + \mu \underline{x}_a[n]\right)^T \underline{x}_a[n]\right)\right]$$

$$= \underline{w}[n-1] + \mu\, \underline{x}_a[n]\left[1 - u\left(\underline{w}^T[n-1]\underline{x}_a[n] + \mu\, \underline{x}_a^T[n]\underline{x}_a[n]\right)\right]$$

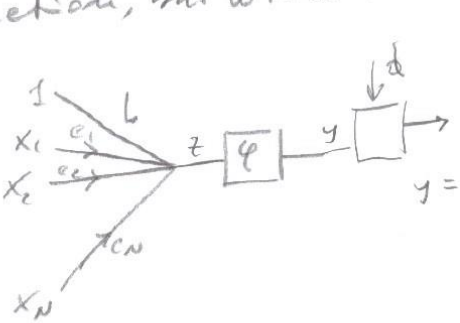
Now since $\mu>0$ and $\underline{x}_a^T[n]\,\underline{x}_a[n]>0$, the update tends to push $u(\cdot)\Rightarrow 1$, i.e. towards a correct classification. The dual situation happens when $d[n]=0$ and $y[n]=1$.

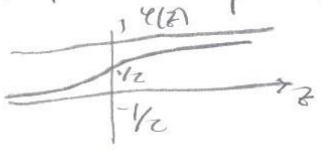
A standard gradient algorithm here cannot be derived because $u(\cdot)$ is not differentiable. We will see in the following how we can find the perceptron (criterion) with a smooth sigmoidal function algorithm as the limit of a

BINARY CLASSIFIER WITH A SMOOTH SIGMOIDAL FUNCTION

We consider now the same network structure of the previous section, but with a smooth activation function

$$\underline{x}_a = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} \qquad \underline{w} = \begin{bmatrix} b \\ c_1 \\ \vdots \\ c_N \end{bmatrix}$$





$$y = \varphi(\underline{c}^T \underline{x} + b) = \varphi(\underline{w}^T \underline{x}_a)$$

We have already encountered this architecture when we considered the model-based solution for gaussian and exponential family generative models.
We found that this network with $\varphi(z) = \frac{1}{1+e^{-z}}$ (logistic function) provides the posterior probability for one of the two classes. Recall that the posterior for the other class is the complement to 1 of the output $(1-y)$.
In this case we postulate the above architecture and want to learn its parameters from a training set $\mathcal{T} = \{(\underline{x}[n], d[n]),\ n = 1, \dots, n_2\}$ where the desired output $d[n]$ is either 1 or 0 for the two classes.
We can adopt a standard cost function, such as the sum of squared errors

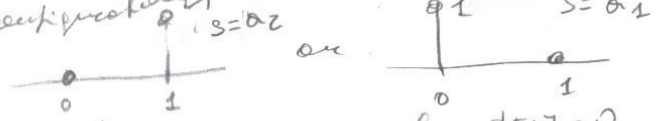
$$\mathcal{E}_{MS}(\underline{w}) = \frac{1}{n_2} \sum_{n=1}^{n_2} \psi(d[n] - \varphi(\underline{w}^T \underline{x}_a[n])) = \frac{1}{n_2} \sum_{n=1}^{n_2} (d[n] - \varphi(\underline{w}^T \underline{x}_a[n]))^2$$

or more appropriately a binary cross-entropy (discussed in a previous section)

$$\mathcal{E}_{ce}(\underline{w}) = \frac{1}{n_2} \sum_{n=1}^{n_2} H\left(\underline{d}[n], \varphi(\underline{w}^T \underline{x}[n])\right)$$

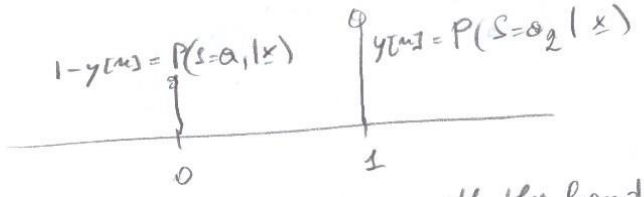
$$= \frac{1}{n_2} \sum_{n=1}^{n_2} \left[d[n] \log \frac{1}{\varphi(\underline{w}^T \underline{x}[n])} + (1 - d[n]) \log \frac{1}{1 - \varphi(\underline{w}^T \underline{x}[n])} \right]$$

The cross-entropy appears to be a more appropriate choice if we want to attribute to the output of the classifier the meaning of a posterior probability. Indeed we can see the desired value $d[n]$ as the "desired distribution" with the two configurations

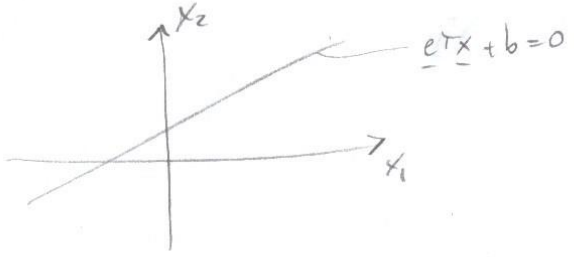


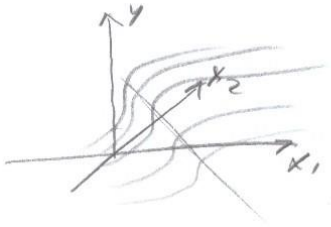
for $d[n]=1$ for $d[n]=0$

and the output being the posterior distribution



The network, just as in the case with the hard threshold, bases its decision on a single hyperplane that divides the input space in two regions. The difference





here is that there is a smooth transition between the two regions (smooth ridge).

Even if the cross-entropy is the most appropriate loss function for this case, the squared error produces similar results. Therefore for completeness we consider both.

GRADIENT WITH SQUARE LOSS

The gradient with the square loss is

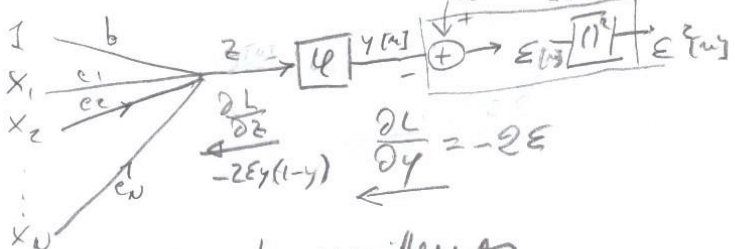
$$\nabla_w \mathcal{E}_{ms}(\underline{w}) = -\frac{2}{n_2} \sum_{n=1}^{n_2} \left(d[n] - \varphi(\underline{w}^T \underline{x}_a[n])\right) \varphi'\left(\underline{w}^T \underline{x}_a[n]\right) \underline{x}_a[n]$$

Using the well-known derivative for the log-sig function

$$\varphi'(\xi) = \varphi(\xi)(1-\varphi(\xi)) \qquad \text{(see Appendix)}$$

$$\nabla_w \mathcal{E}_{ms}(\underline{w}) = -\frac{2}{n_2} \sum_{n=1}^{n_2} \left(d[n] - y[n]\right) y[n](1-y[n]) \underline{x}_a[n]$$

Note the backpropagation flow before, also through the sigmoidal block



The batch iteration can be written as

$$w[K] = w[K-1] + \mu \sum_{n=1}^{n_2} \left(d[n] - y_{K-1}[n]\right) y_{K-1}[n](1-y_{K-1}[n]) \underline{x}_a[n]$$

The notation $y_{K-1}[n]$ is used to emphasize that we use the "old" values $w[K-1]$ to compute the output

$$y_K[n] = \underline{w}^T[K-1] \underline{x}_a[n]$$

We can give now some intuitive interpretation to the updates.

The contribution of the nth example to the previous weight $w_i[k-1]$ is made evident as

$$w_i[k] = w_i[k-1] + \mu \left(d[n] - y_{k-1}[n]\right) y_{k-1}[n] \left(1 - y_{k-1}[n]\right) x_{ai}[n] + \dots \text{(contribution from other examples)} \dots$$

Now suppose that the desired value in the nth example is $d[n] = 1$, but the output $y_{k-1}[n] = \delta$, where δ is a small positive number. We are clearly in an undesirable situation because we would like $y_{k-1}[n]$ to be close to $d[n] = 1$. Let us see if the update helps to correct the situation.

The contribution to the output is

$$w_i[k]\, x_{ai}[n] = w_i[k-1]\, x_{ai}[n] + \mu \underbrace{\left(d[n] - y_{k-1}[n]\right) y_{k-1}[n] \left(1 - y_{k-1}[n]\right) x_{ai}^2[n]}_{\Delta_i} + \dots$$

where the error $d[n] - y_{k-1}[n] = 1 - \delta$ is positive,

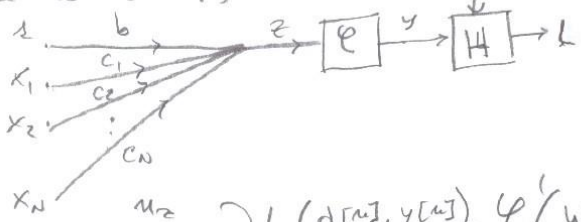
$$y_{k-1}[n]\left(1 - y_{k-1}[n]\right) = \delta\,(1-\delta) > 0$$

is also positive and $x_{ai}[n] > 0$. Therefore the contribution Δ_i to the output is positive and tends to push $y_{k-1}[n]$ towards higher values, i.e. towards $d[n] = 1$.

A dual situation happens when $d[n] = 0$ and $y_{k-1}[n] = 1 - \delta$ (with $\delta > 0$ small number). The error $d[n] - y_{k-1}[n] = 0 - 1 + \delta < 0$ is negative, $y_{k-1}[n]\left(1 - y_{k-1}\right) = (1-\delta)\delta > 0$, $x_{ai}^2[n] > 0$ and the update will be negative pushing the output towards smaller values.

GRADIENT WITH THE CROSS ENTROPY

Using the cross-entropy as loss we have



$$\nabla_{\underline{w}} \mathcal{E}_{ce}(\underline{w}) = \frac{1}{n_z} \sum_{n=1}^{n_z} \frac{\partial L(d[n], y[n])}{\partial y[n]} \varphi'(\underline{w}^T \underline{x}_a[n]) \underline{x}_a[n]$$

$$L(d,y) = H(d,y) = d \log \frac{1}{y} + (1-d) \log \frac{1}{1-y}, \text{ and}$$

$$\frac{\partial L(d,y)}{\partial y} = \frac{y-d}{y(1-y)} \quad \text{(see Appendix)}$$

the gradient is

$$\nabla_{\underline{w}} \mathcal{E}_{ce}(\underline{w}) = \frac{1}{n_z} \sum_{n=1}^{n_z} \left[\frac{y[n] - d[n]}{y[n](1-y[n])} \right] y[n](1-y[n]) \underline{x}_a[n]$$

The batch update for cross-entropy loss is

$$\underline{w}[k] = \underline{w}[k-1] + \mu \sum_{n=1}^{n_z} \left(d[n] - y_{k-1}[n] \right) \underline{x}_a[n]$$

(Analogous similarity with the perceptron rule)

where again $y_{k-1}[n] = \underline{w}^T[k-1] \underline{x}_a[n]$ is the output computed with the "old values" $\underline{w}[k-1]$.

Also here it is quite interesting to give an intuitive interpretation to the update by looking at the contribution of the nth example to the update of the ith weight and its impact on the output.

The contribution to the i-th weight could evidenced as

$$w_i[k] = w_i[k-1] + \mu\left[d[n] - y_{k-1}[n]\right] x_{ni} + \text{...contributions from other examples}$$

and its contribution to the output z

$$w_i[k]\, x_{ni}[n] = w_i[k-1]\, x_{ni}[n] + \mu\left[d[n] - y_{k-1}[n]\right] x_{ni}^2 + \text{...contribution from other examples...}$$

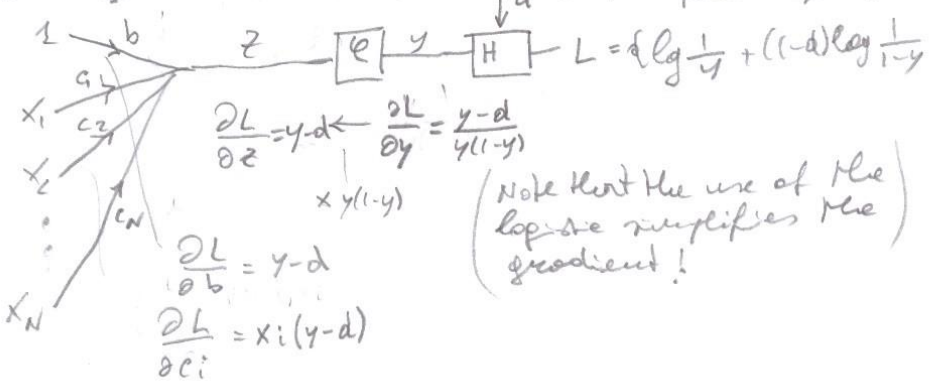
Now if $d[n]=1$ and $y_{k-1}[n]=\delta$ with $\delta > 0$ small value, we are in an undesirable situation because we would like to have $y_{k-1}[n]$ close to one.
The correction term is

$$\mu\left(d[n] - y_{k-1}[n]\right) x_{ni}^2[n] = \mu(1-\delta) x_{ni}^2 > 0$$

the pushed the output z and then y to higher values.
The dual situation is when $d[n]=0$ and $y_{k-1}[n]=1-\delta$.
Again undesirable because we would like to have $y_{k-1}[n]$ be close to zero. The correction term is

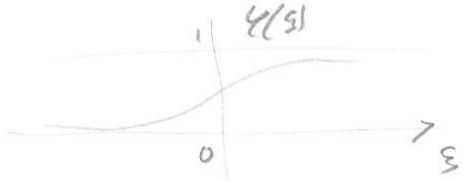
$$\mu\left(d[n] - y_{k-1}[n]\right) x_{ni}^2[n] = \mu\left(0-1+\delta\right) x_{ni}^2[n] < 0$$

and the output z and then y is pushed towards lower values.
The backpropagation flow is also interesting.
On a generic pair $(x[n], d[n])$ (drop n for simplicity)



BINARY SMOOTH CLASSIFIERS WITH OTHER SIGMOIDAL FUNCTIONS

We have assumed in the previous sections that the smooth sigmoid placed at the output of the classifier is a logistic function. This choice was the consequence of the results on model-based classifiers where we had knowledge of the likelihoods (i.e. features are exponential family). Certainly special densities may be hidden in the data, but we generally do not know much in data-driven methods.
Therefore any sigmoid-shaped function may be used

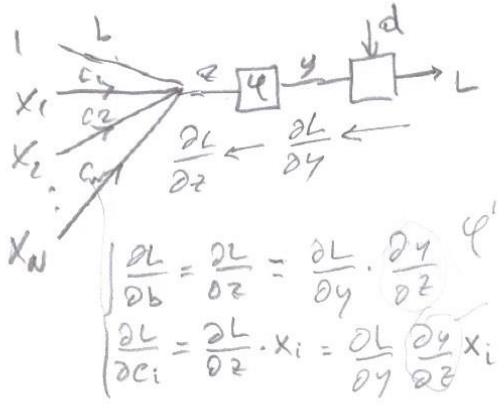


It must be positive, non decreasing and such that $\varphi(s) \xrightarrow[s \to \infty]{} 1$ and $\varphi(s) \xrightarrow[s \to -\infty]{} 0$

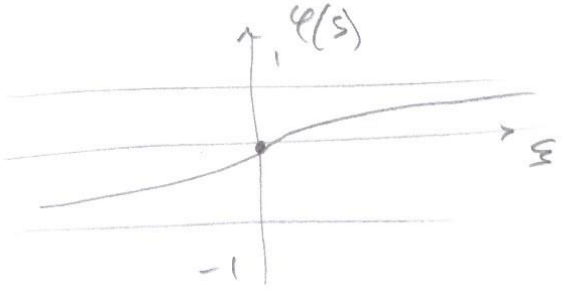
Recall that this is the behaviour of any cumulative distribution functions (cdf) where its derivative is the probability density function (pdf)

Therefore we could pick any (pdf, cdf) pair we know, such as (Uniform, piecewise linear), (gaussian, cumulative gaussian), (Laplacian, Laplacian cdf)
Various choices are detailed in the Appendix on Activation functions.

In all cases we backpropagate the loss in the sigmoidal function φ (LC.12)



In some cases the binary classifier can be slightly changed with target values for the two classes ~~for the output~~ $d \in \{-1, 1\}$, instead of $d \in \{0, 1\}$.
The sigmoidal function must be scaled and shifted



and be such that $\varphi(s) \underset{s \to \infty}{\to} 1$ and $\varphi(s) \underset{s \to -\infty}{\to} -1$.
Examples of these sigmoids are the hyperbolic tangent and the trigonometric arctangent described in the Appendix on activation functions.

To interpret the output of the sigmoid as a posterior we must scale and shift to get

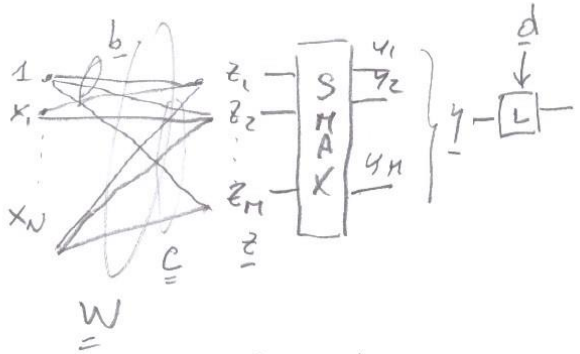
$$P_r\{class\ 1 | x\} = \frac{y+1}{2}$$

or the inverse

$$y = 2P(class\ 1|x) - 1$$

MULTI-CLASS LINEAR CLASSIFIER

The typical multi-class linear classifier has the following architecture



$$\underline{z} = \underline{\underline{C}}^T \underline{x} + \underline{b} \qquad \underline{y} = g_{max}(\underline{z})$$

Using the augmented vector $\underline{x}_a = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} 1 \\ \underline{x} \end{bmatrix}$ and the augmented matrix $\underline{\underline{W}} = \begin{bmatrix} \underline{b}^T \\ \underline{\underline{C}} \end{bmatrix}$

we have $\underline{z} = \underline{\underline{W}}^T \underline{x}_a$, $\underline{y} = g_{max}(\underline{z})$

[In note (*) we explode the augmented matrix and vector for better visualization of indices.]

$g_{max}(z)$ is the usual softmax function already encountered in model-based architectures for M-ary classifiers. The linear part comes out from the model if we assume gaussian (or exponential family) classes with the same covariance matrix.

Here, instead, we have no knowledge of the generative model. The structure is postulated and we have to learn it from the training set

$$\mathcal{T} = \{(\underline{x}_a[n], \underline{d}[n]), n = 1, \dots, n_z\}$$

(*) NOTE For better visualization of the indexes in the matrices, let us explode them.

$\underline{\underline{C}}$ and $\underline{b}$ are defined as

$$\underline{\underline{C}} = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1M} \\ c_{21} & c_{22} & & c_{2M} \\ \vdots & & & \\ c_{N1} & c_{N2} & \cdots & c_{NM} \end{bmatrix} \qquad \underline{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix}$$

$\underline{\underline{C}}$: $N \times M$ $\qquad$ $\underline{b}$: M

The output $\underline{z}$ is computed as

$$\underline{z} = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_M \end{bmatrix} = \underline{\underline{C}}^T \underline{x} + \underline{b} = \begin{bmatrix} c_{11} & c_{21} & \cdots & c_{N1} \\ c_{12} & c_{22} & & c_{N2} \\ & & & \\ c_{1M} & c_{2M} & \cdots & c_{NM} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_M \end{bmatrix}$$

note (*)

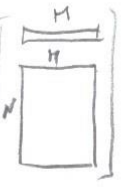
Therefore c_{ij} connects input x_i with output z_j

In the augmented representation

$$\underline{z} = \begin{bmatrix} b_1 & c_{11} & c_{21} & \cdots & c_{N1} \\ b_2 & c_{12} & c_{22} & \cdots & c_{N2} \\ & & & & \\ b_M & c_{1M} & c_{2M} & \cdots & c_{NM} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} \underline{b} & \underline{\underline{C}}^T \end{bmatrix} \underline{x}_a = \underline{\underline{W}}^T \underline{x}_a$$

$\underline{\underline{W}}^T$: $M \times (N+1)$ $\qquad$ $\underline{\underline{W}}$: $(N+1) \times M$

$$\underline{\underline{W}} = \begin{bmatrix} \underline{b}^T \\ \underline{\underline{C}} \end{bmatrix}$$



For convenience, define a $\underline{\underline{W}}$ with a row index that starts from zero (Lec.15)

$$\underline{\underline{W}} = \begin{bmatrix} \underline{b}^T \\ \underline{\underline{C}} \end{bmatrix} \begin{matrix} \} \\ \} \end{matrix} \begin{matrix} 1 \\ N \end{matrix} \quad \begin{matrix} \\ N+1 \end{matrix}$$

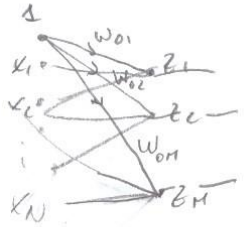
$$= \begin{bmatrix} w_{01} & w_{02} & \cdots & w_{0M} \\ w_{11} & w_{12} & \cdots & w_{1M} \\ w_{21} & w_{22} & \cdots & w_{2M} \\ \vdots & & & \\ w_{N1} & w_{N2} & \cdots & w_{NM} \end{bmatrix}$$

$$\underline{z} = \underline{\underline{W}}^T \underline{x}_a = \begin{bmatrix} w_{01} & w_{11} & w_{21} & \cdots & w_{N1} \\ w_{02} & w_{12} & w_{22} & & w_{N2} \\ \vdots & & & & \\ w_{0M} & w_{1M} & w_{2M} & & w_{NM} \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

note(*)

Therefore w_{0j} connects 1 to z_j

w_{ij} connects x_i to z_j



end of note(*)

As explained in one of the previous sections, $\underline{d}[n]$ are distribution vectors, typically one-hot

$$\underline{d} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ for } a_1 \quad \underline{d} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \text{ for } a_2 \quad \cdots \quad \underline{d} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \text{ for } a_M$$

We already pointed out that, more generally, $\underline{d}[n]$ may be a distribution expressing uncertainty about the class in the training set. For example when we are not sure about the label and we have two concurring classes with the same probability, we could use a 2-hot distribution with $1/2$ in the 2 corresponding positions. More generally the training set may not be "sharp" and provide only a distribution on some samples.

The problem is formulated as

$$\underline{W}^* = \arg\min_{\underline{W}} \mathcal{E}(\underline{W}) = \arg\min_{\underline{W}} \frac{1}{N_{TR}} \sum_{n=1}^{N_{TR}} L(\underline{d}[n], \underline{y}[n])$$

As usual we seek the solution with an iterative update that requires the computation of the gradient

$$\nabla_{\underline{W}} \mathcal{E}(\underline{W}) = \left[\frac{\partial \mathcal{E}}{\partial w_{ij}} \right]_{\substack{i=0,\dots,N \\ j=1,\dots,M}}$$

Note that the gradient is a $(N+1) \times M$ matrix (gradient flow)

More specifically

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \sum_{\ell=1}^{M} \frac{\partial \mathcal{E}}{\partial z_\ell} \frac{\partial z_\ell}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}}$$

because only z_j depends on w_{ij}. Therefore

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial z_j} x_{ai}$$

(Recall that $x_{a0} = 1$ and corresponds to the connection that takes care of the bias)

Now to compute $\frac{\partial E}{\partial z_j}$, $j=1,\dots,M$, we need to define a loss function that measures the discrepancy between $\underline{y}$ and $\underline{d}$. Any loss function could be used such as those used for regression. The main one being obviously the sum of squares $L(d,y)=\sum_{j=1}^{M}(d_j-y_j)^2$. However since both y and $\underline{d}$ are distributions, the most appropriate choice would be the cross-entropy $L(\underline{d},\underline{y})=\sum_{j=1}^{M} d_j \ln\frac{1}{y_j}$. For completeness, just as we did for the binary classifier, we consider both the sum of squares and the cross-entropy.

LINEAR MULTICLASS WITH THE SQUARE LOSS

The cost function to minimize using the square loss is

$$E_{ms}(\underline{W})=\frac{1}{N_2}\sum_{n=1}^{N_2}\sum_{k=1}^{M}\left(d_k^{[n]}-y_k^{[n]}\right)^2$$

As shown in the previous section, we need

$$\frac{\partial E_{ms}}{\partial z_j},\quad j=1,\dots,M$$

which can be written as

$$\frac{\partial E_{ms}}{\partial z_j}=-\frac{1}{N_2}\sum_{n=1}^{N_2}\sum_{k=1}^{M}2\left(d_k^{[n]}-y_k^{[n]}\right)\frac{\partial y_k}{\partial z_j}$$

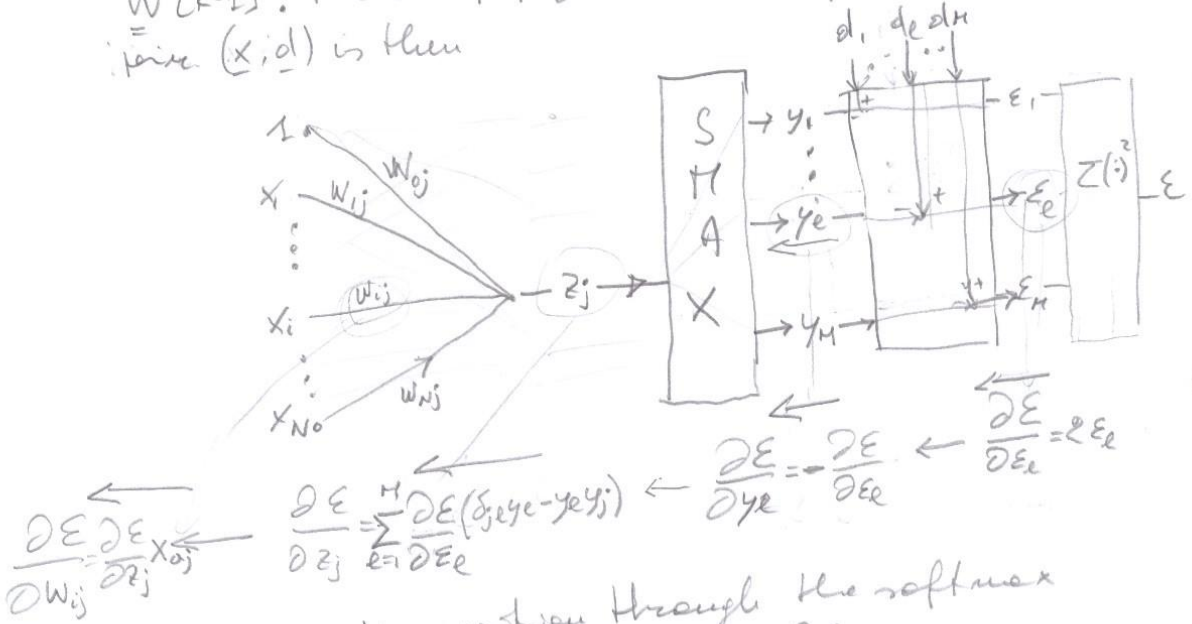
The derivative $\frac{\partial y_k}{\partial z_j}$ is computed across the softmax function and is (see Appendix)

$$\frac{\partial y_k}{\partial z_j}=\delta_{jk}y_k-y_k y_j$$

Then the batch update for the generic weight is

$$W_{ij}[k] = W_{ij}[k-1] + \mu \frac{1}{N_z} \sum_{n=1}^{N_z} \sum_{\ell=1}^{M} \left(d_\ell[n] - y_\ell[n]\right) \left(\delta_{j\ell}\, y_\ell[n] - y_\ell[n]\, y_j[n]\right)$$

Where as usual the outputs $y_i[n]$ are computed using $W[k-1]$. The backpropagation on a single example pair $(\underline{x}; \underline{d})$ is then



Note that the backpropagation through the softmax requires on each z_j all the derivatives $\frac{\partial \varepsilon}{\partial y_\ell}$ $\ell = 1, \dots, M$.

LINEAR MULTICLASS WITH CROSS-ENTROPY

The cost function to minimize using the cross-entropy loss is

$$\mathcal{E}_{ce}(\underline{w}) = \frac{1}{n_z} \sum_{n=1}^{n_z} \sum_{m=1}^{M} d_m^{[n]} \lg \frac{1}{y_m^{[n]}}$$

We need as before

$$\frac{\partial \mathcal{E}_{ce}}{\partial z_j}, \quad j = 1, \ldots, M$$

$$\frac{\partial \mathcal{E}_{ce}}{\partial z_j} = \sum_{\ell=1}^{M} \frac{\partial \mathcal{E}_{ce}}{\partial y_\ell} \frac{\partial y_\ell}{\partial z_j}$$

$$\frac{\partial \mathcal{E}_{ce}}{\partial y_\ell} = \frac{\partial}{\partial y_\ell} \sum_{m=1}^{M} d_m \lg \frac{1}{y_m} = -\frac{\partial}{\partial y_\ell} d_\ell \lg y_\ell = -\frac{d_\ell}{y_\ell}$$

Using the derivative across the softmax, we

$$\frac{\partial y_\ell}{\partial z_j} = \delta_{j\ell} y_\ell - y_\ell y_j$$

we have

$$\frac{\partial \mathcal{E}_{ce}}{\partial z_j} = \sum_{\ell=1}^{M} -\frac{d_\ell}{y_\ell} \cdot (\delta_{j\ell} y_\ell - y_\ell y_j) =$$

$$= -\sum_{\ell=1}^{M} d_\ell \delta_{j\ell} + \underbrace{\sum_{\ell=1}^{M} d_\ell}_{=1} y_j$$

$$= -d_j + y_j$$

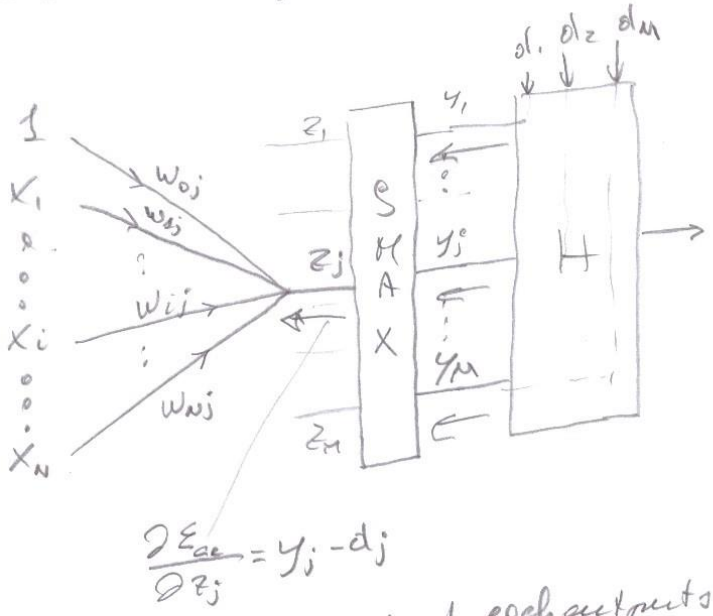
The derivative with respect to the generic weight w_{ij} is

$$\frac{\partial \mathcal{E}_{ce}}{\partial w_{ij}} = \frac{\partial \mathcal{E}}{\partial z_j} x_{ai} = (y_j - d_j) x_{ai}$$

The gradient batch update is then

$$W_{ij}[k] = W_{ij}[k-1] + \mu \sum_{m=1}^{M_2} (d_j[m] - y_j[m]) x_{ai}[m] \quad \forall i,j$$

where as usual $y_j[m]$ is computed using the "old" weights $\underline{\underline{W}}[k-1]$.



$$\frac{\partial \mathcal{E}_{ce}}{\partial z_j} = y_j - d_j$$

Note that despite the fact that each outputs of the softmax y_j depend on all the inputs $z_1 \cdots z_M$, the derivative using the cross-entropy simplifies greatly the backpropogation through both blocks.

To dig deeper into the updating equation, assume as usual that only one of the d_j is equal to one (we pointed out above that this is generalizable to any distribution on $\underline{d}$). Say that $d_q = 1$ and the others are zero. The weights converging on z_j will be updated according to the following simple rules

$$\begin{cases} \Delta W_{ij} = \mu (1 - y_q) x_{ai} & \text{for } j = q \\ \Delta W_{ij} = -\mu\, y_q\, x_{ai} & \text{for } j \neq q \end{cases}$$

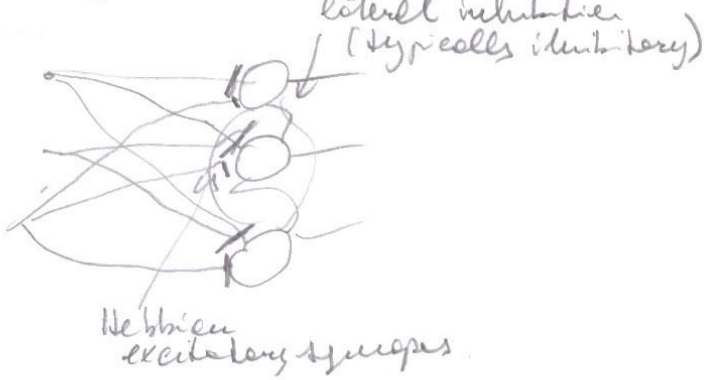
The variation on w_j caused by this update at a generic time step is

$$\Delta w_{ij} x_i = \begin{cases} \mu (1 - y_q) x_{ia}^2 & \text{for } j = q \\ -\mu y_j x_{ai}^2 & \text{for } j \neq q \end{cases}$$

The gradient algorithm pushes up the desired winner y_q because $(1 - y_q) x_{ia}^2 > 0$. Conversely the other softmax outputs are pushed down as $-y_q x_{ai}^2 < 0$ for $j \neq q$.

OBSERVATION (Hebbian learning)

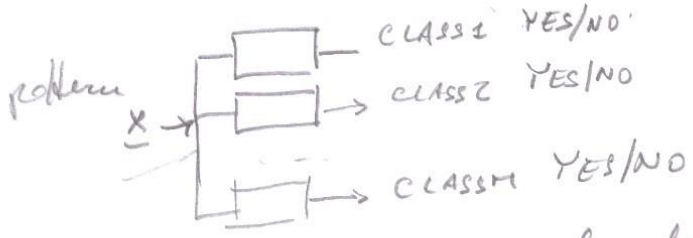
In neurobiology one of the fundamental assumptions about neural adaptation is that synapses become stronger if they succeed in moving the target neuron to fire. Viceversa when they do not succeed they tend to retreat (Hebbian hypothesis). Clearly there is no biological neuron here, but one analogy can be drawn because the competition that happens in the softmax block does resemble a lateral inhibition with a winning firing neuron.



LEZZ

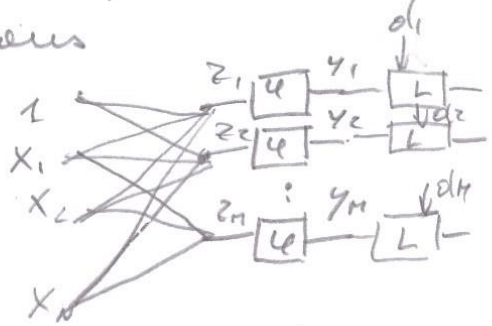
MULTIPLE BINARY CLASSIFIERS

In discriminating among M classes, instead of using a single multi-class decision maker, we could think of M binary classifiers that tell us if a pattern belongs to a class or not.



In this architecture there is clearly no "competition" among the classifiers that simply perform a binary decision, or more generally provide a posterior probability on one of the two responses (the other is the complement to one)

A neural architecture with M binary linear classifier is a parallel of M perceptrons

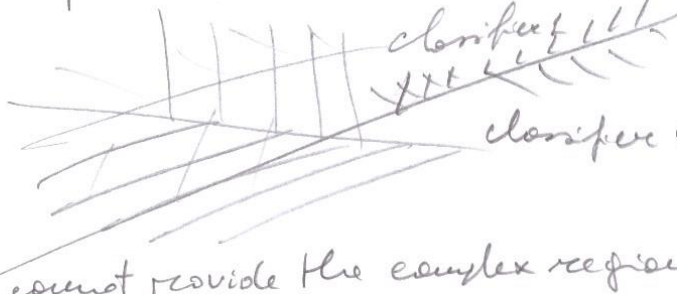


We have already discussed the linear binary classifier where g may be the logistic function and the loss L may be one of many. The desired inputs can be as usual organized in one-hot vectors

$$\underline{d} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad \underline{d} = \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad \cdots \quad \underline{d} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

are use different configurations.

Note that linear classifiers can only divide the input space in two parts



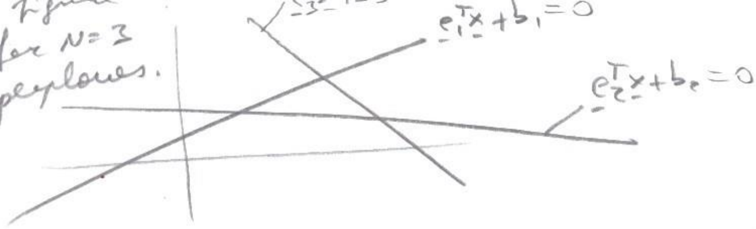
must cannot provide the complex regions that are obtained using the softmax. We will provide an explanation in the following.

HOW THE LINEAR CLASSIFIERS WORK

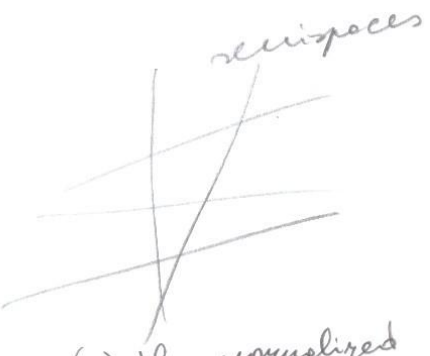
In both (*) and (**) we have M affine transformations

$$z_i = \underline{c}_i^T x + b_i \qquad i = 1, \dots, M$$

that correspond to hyperplanes in the space of $\underline{x}$. The figure shows for N=3 M=3 hyperplanes.



Now each z_i is the unnormalized distance(*) from hyperplane i, with a positive value on one side and negative on the other.
In the set of linear classifiers, the M hyperplanes just divide the space in two semi-spaces, while in the architecture with the softmax they "compete" and can give rise to more complex regions.



linclass.m

(*) the normalized distance would be

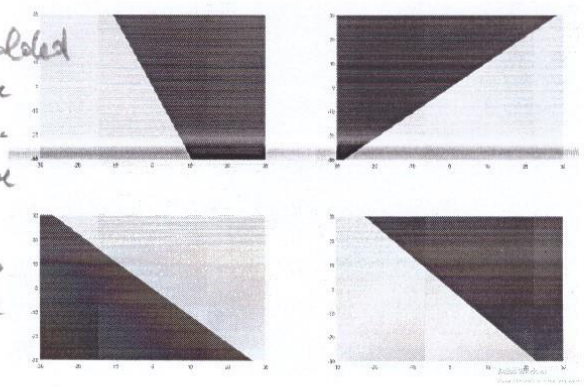
$$\frac{c_i^T x + b_i}{\sqrt{\| c_i \|_2}}$$

far vedere anche la figura.
con le rette sottile e le traiettorie
e maggiori hard e soft per ruole diversi

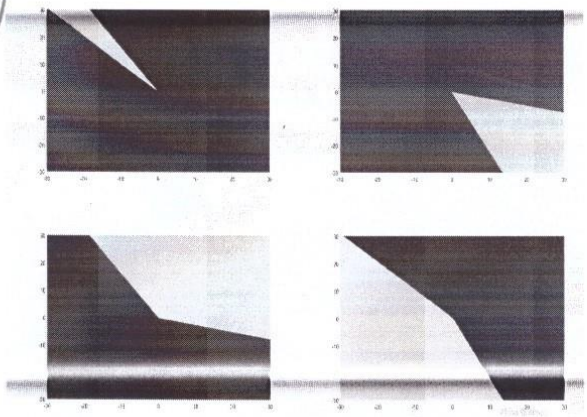
In this following picture we display the regions for random (c_i, b_i) $i=1,..,4$ for 4 binary classifiers and for the softmax.

LC.25

(we have thresholded the values for visualization. Recall that the smoothness of the transitions is governed by the magnitude of z_i. Large z_i's correspond to sharp transitions in both cases)



4 binary classifiers



Decision regions from a softmax $M=4$

Note that the boundaries of the softmax regions are not necessarily the hyperplanes seen in the binary decisions. The competition arising from the unnormalized distances from the hyperplanes move the boundaries in different positions.

Therefore the architecture with the softmax function is much more powerful that the parallel of perceptrons in shaping the decision regions. The structure with multiple binary perceptrons is usually inserted in multi-layer architectures.