EE 649 Pattern Recognition

Neural Networks

Ulisses Braga-Neto

ECE Department

Texas A&M University

Origins: Kolmogorov-Lorentz Theorem

Every continuous function $f : [0,1]^d \rightarrow R$ can be written as:

$$f(x) = \sum_{i=1}^{2d+1} F_i \left(\sum_{j=1}^{d} G_{ij}(x_j) \right)$$

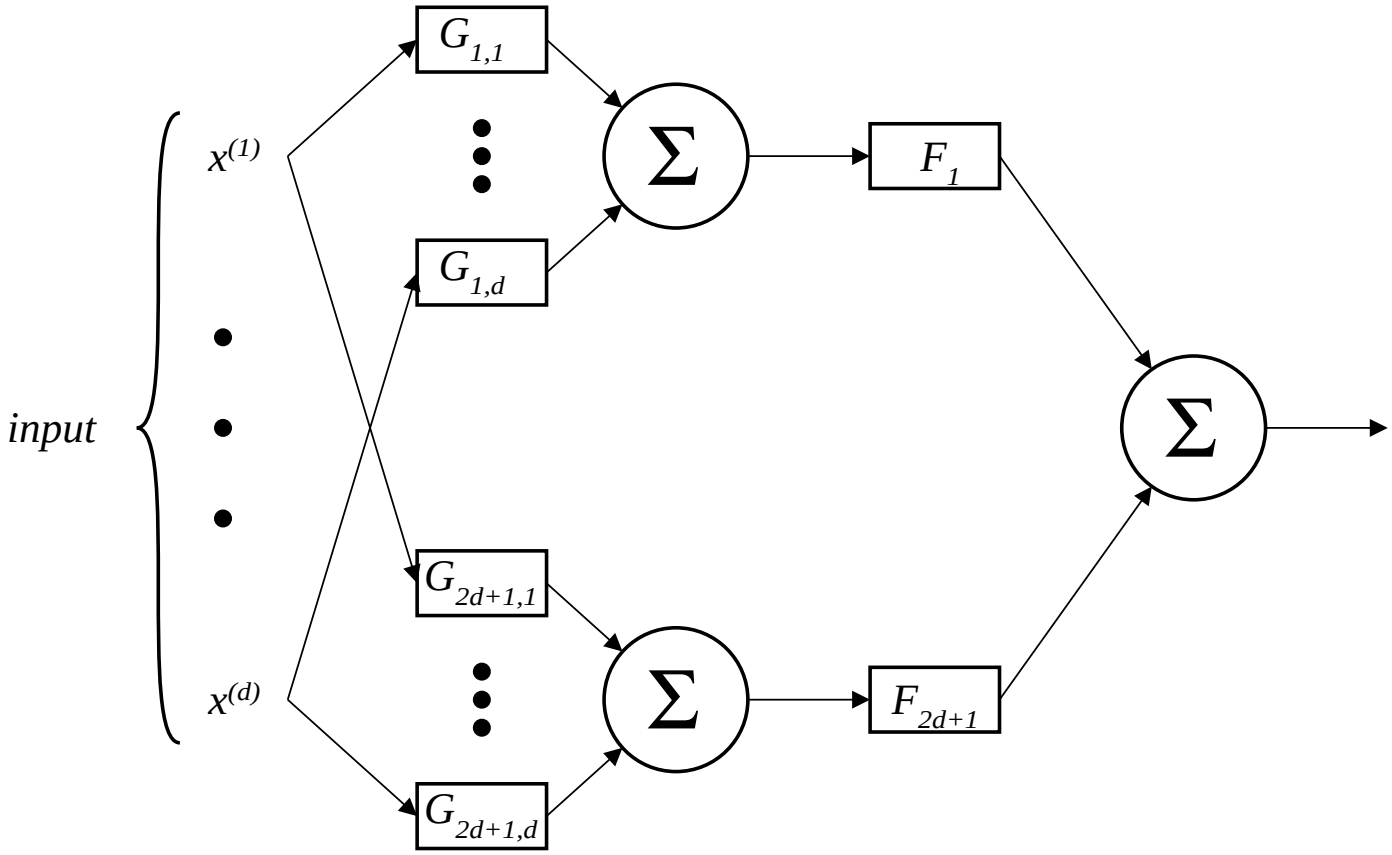
where the $F_i : R \rightarrow R$ and $G_{ij} : R \rightarrow R$ are continuous functions. This means that a multivariate function can be computed as a finite sum of univariate functions of sums of univariate functions of the coordinates.

For example, for $d = 2$, the function $f : [0,1]^2 \rightarrow R$, given by $f(x,y) = xy$ can be written as:

$$f(x,y) = \frac{1}{4}\left((x+y)^2 - (x-y)^2\right)$$

Kolomogorov-Lorentz Network

This can be represented by means of a *two-layer* network:



Main Idea

It is not obvious at all how to find functions F_i and G_{ij} required to compute exactly a general function f.

Instead, it is possible to use simpler functions:

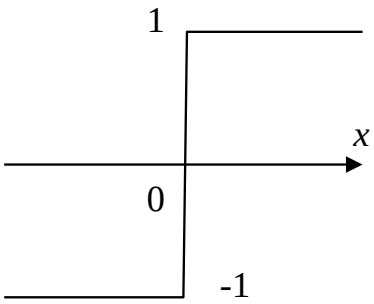
- Linear functions $f(x) = a^T x + a_0 = \sum_{i=1}^{d} a_i x_i + a_0$
- *Sigmoids*: non-decreasing functions $\sigma(x)$ with $\sigma(-\infty) = -1$ and $\sigma(\infty) = 1$. These are the *nonlinearities* that allow neural networks to be nonlinear classifiers.

and obtain arbitrary approximation capability.

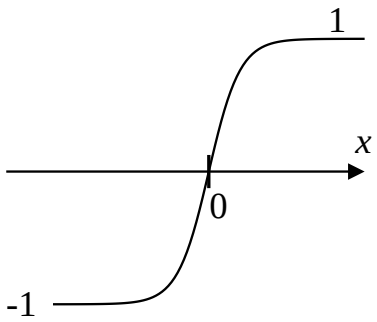
Examples of Sigmoids

The most common nonlinearities (sigmoids) used in neural networks are depicted below.

Threshold

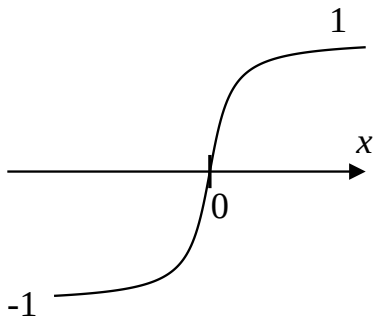


Standard (logistic)



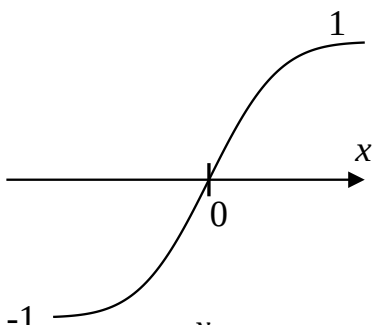
$$\sigma(x) = \frac{1-e^{-x}}{1+e^{-x}}$$

Arctan



$$\sigma(x) = \frac{2}{\pi}\arctan(x)$$

Gaussian



$$\sigma(x) = \sqrt{\frac{2}{\pi}} \int_{-\infty}^{x} e^{-v^2/2}\, dv - 1$$

Feed-Forward Neural Network

A two-layer network with k *hidden neurons* is specified by:

$$\zeta(x) = c_0 + \sum_{i=1}^{k} c_i \sigma(\phi_i(x)) = c_0 + c^T \xi(x)$$

where $\xi(x) = (\sigma(\phi_1(x)), \ldots, \sigma(\phi_d(x)))^T$, with

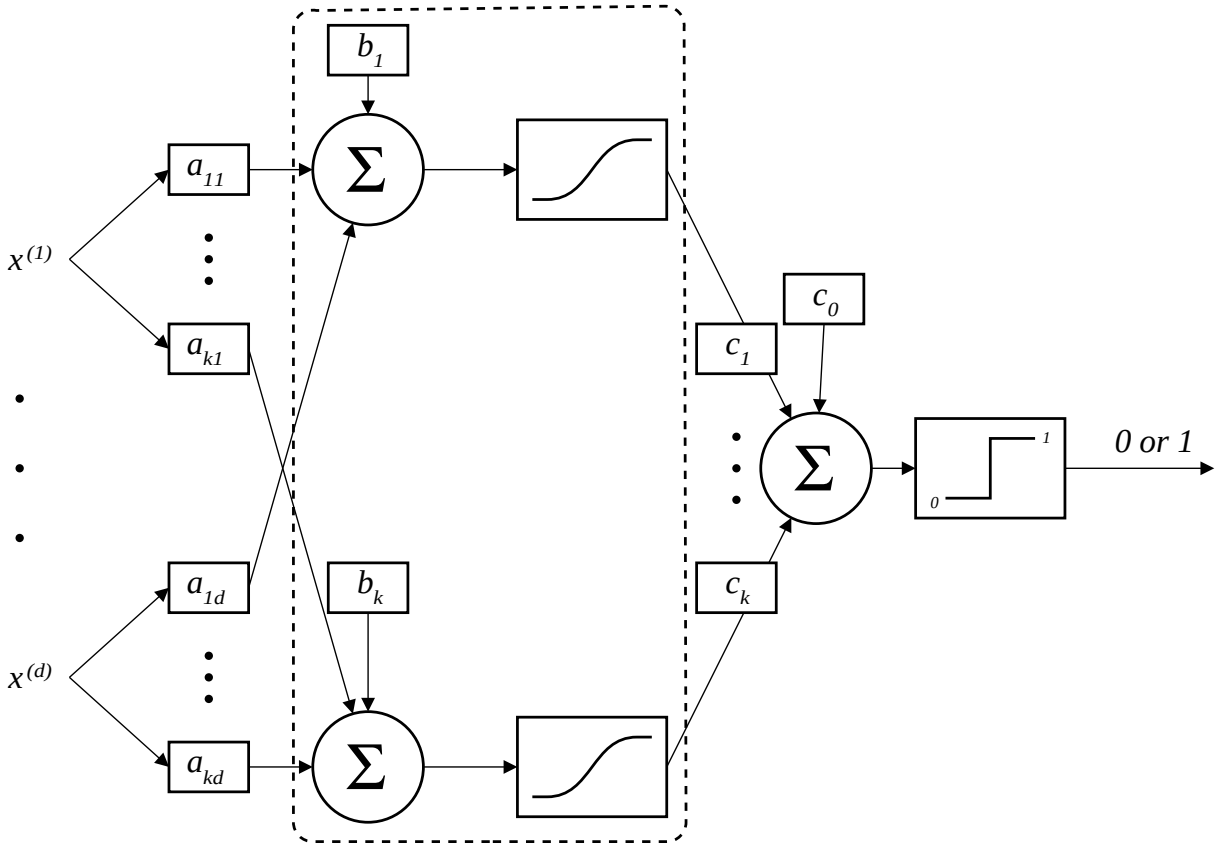
$$\phi_i(x) = b_i + \sum_{j=1}^{d} a_{ij} x_j$$

A neural network *classifier* can be obtained by thresholding the output:

$$\psi(x) = \begin{cases} 1, & \zeta(x) > 0 \\ 0, & \text{otw} \end{cases}$$

Network With One Hidden Layer

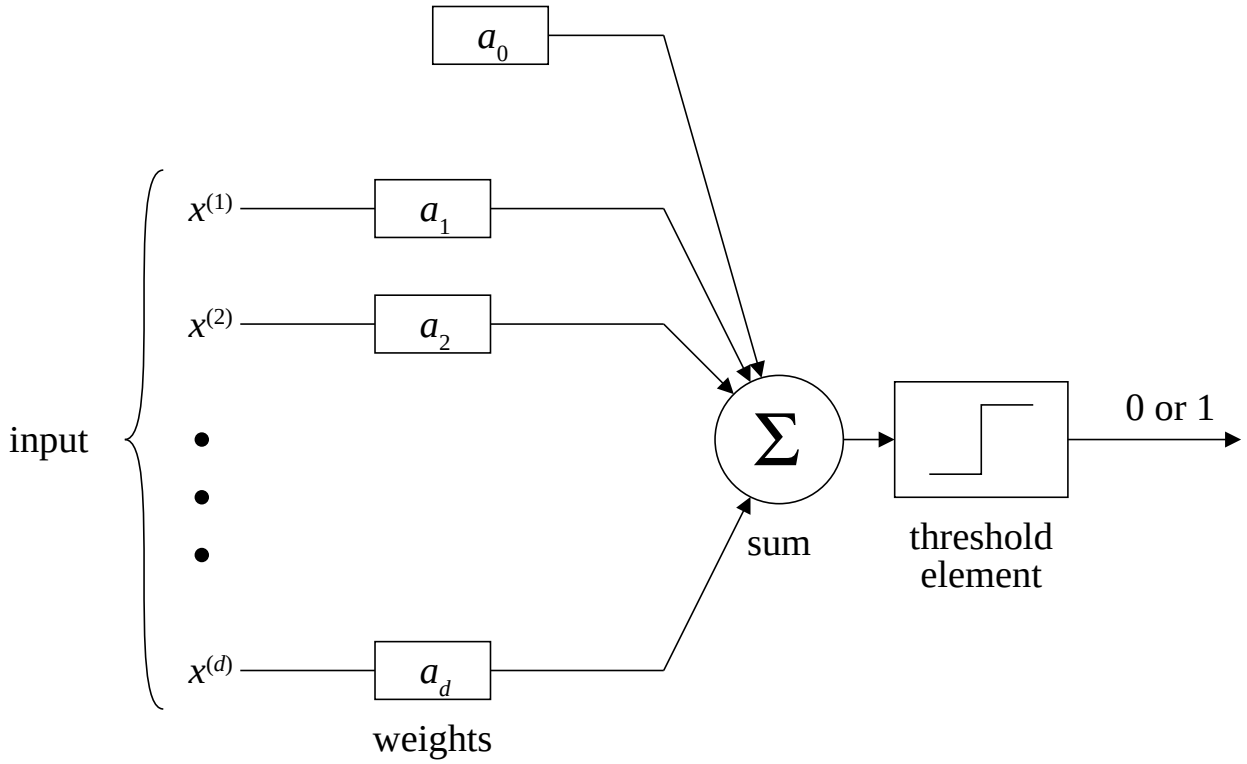
This can be computed by means of a two-layer network:



The dashed box indicates the *hidden layer*.

The Perceptron

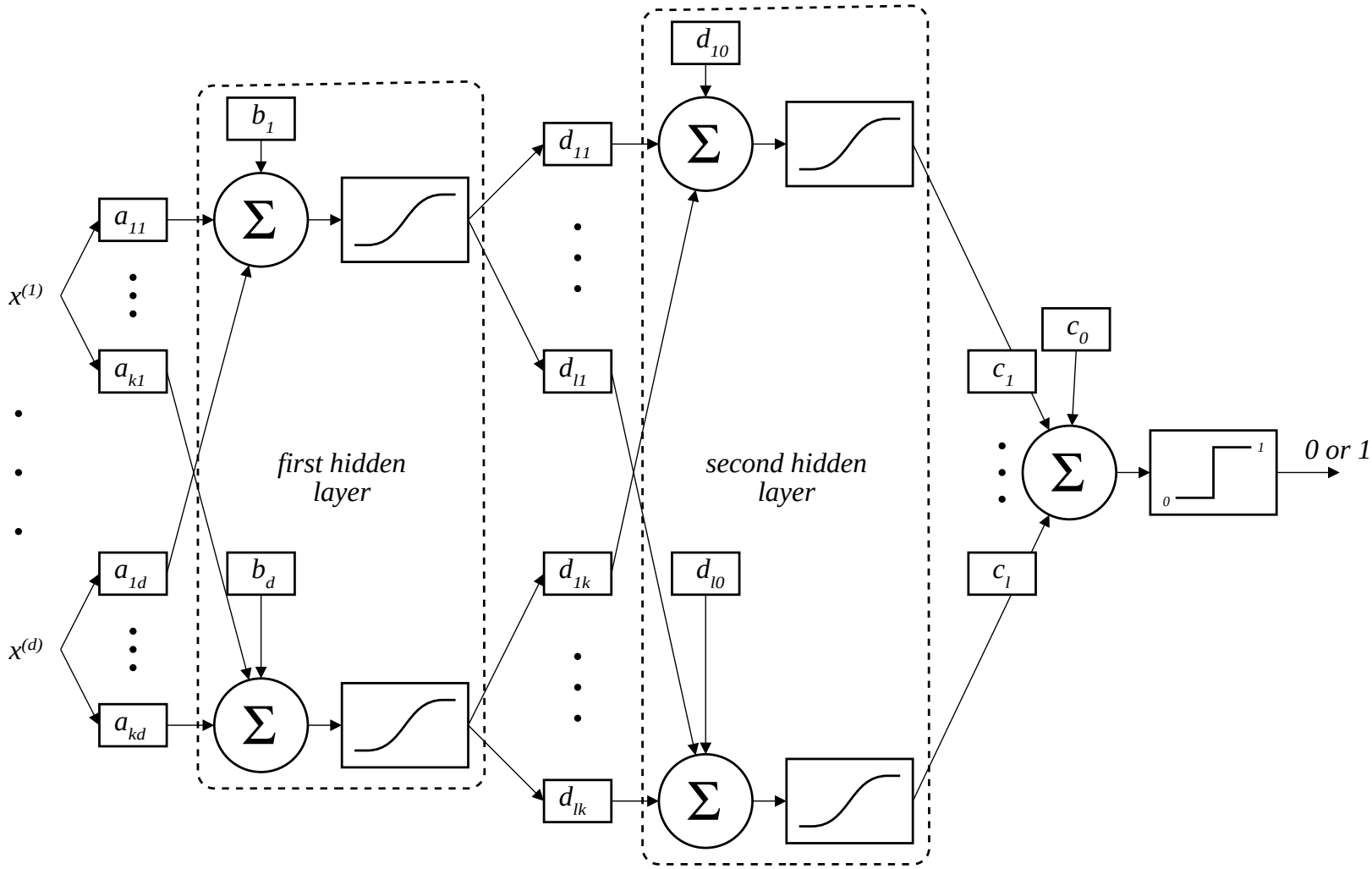
The previous network generalizes the perceptron:



Therefore, neural networks are also called *Multilayer Perceptrons*.

Network with Two Hidden Layers

This idea can be extended to any number of hidden layers:



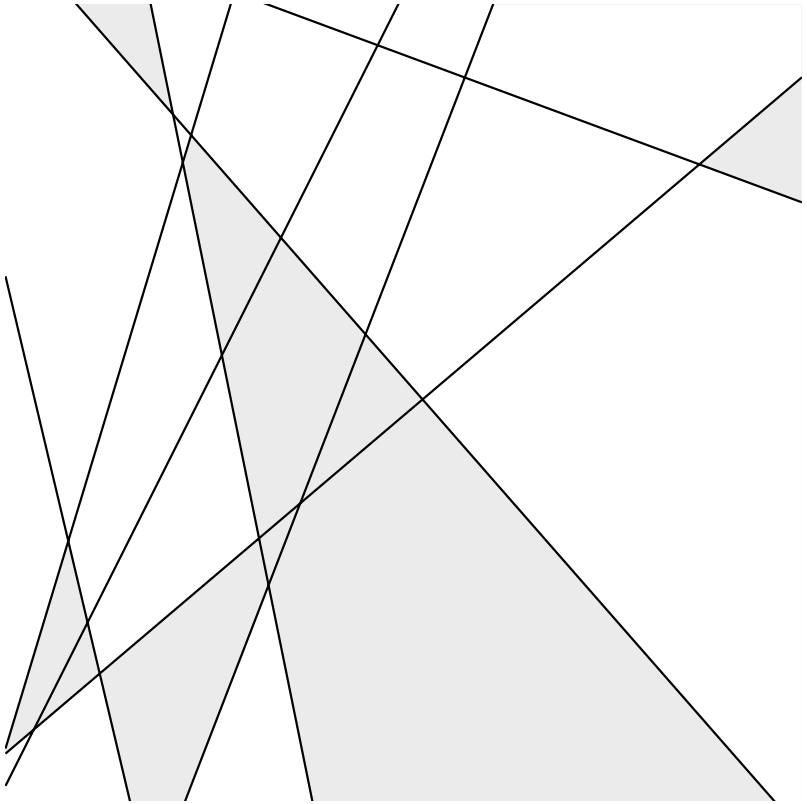
Some Observations

- The neural network approach is similar to the non-linear SVM approach. The hidden layers nonlinearly map the original feature space into a different space, and the output layer acts on the transformed features by means of a linear decision (hyperplane). The decision in the original feature space is nonlinear.

- If the *first* hidden layer is composed of threshold sigmoids, then no matter what follows, the decision regions will be piecewise linear, given by the intersection of k hyperplanes (this is called an *arrangement classifier*.

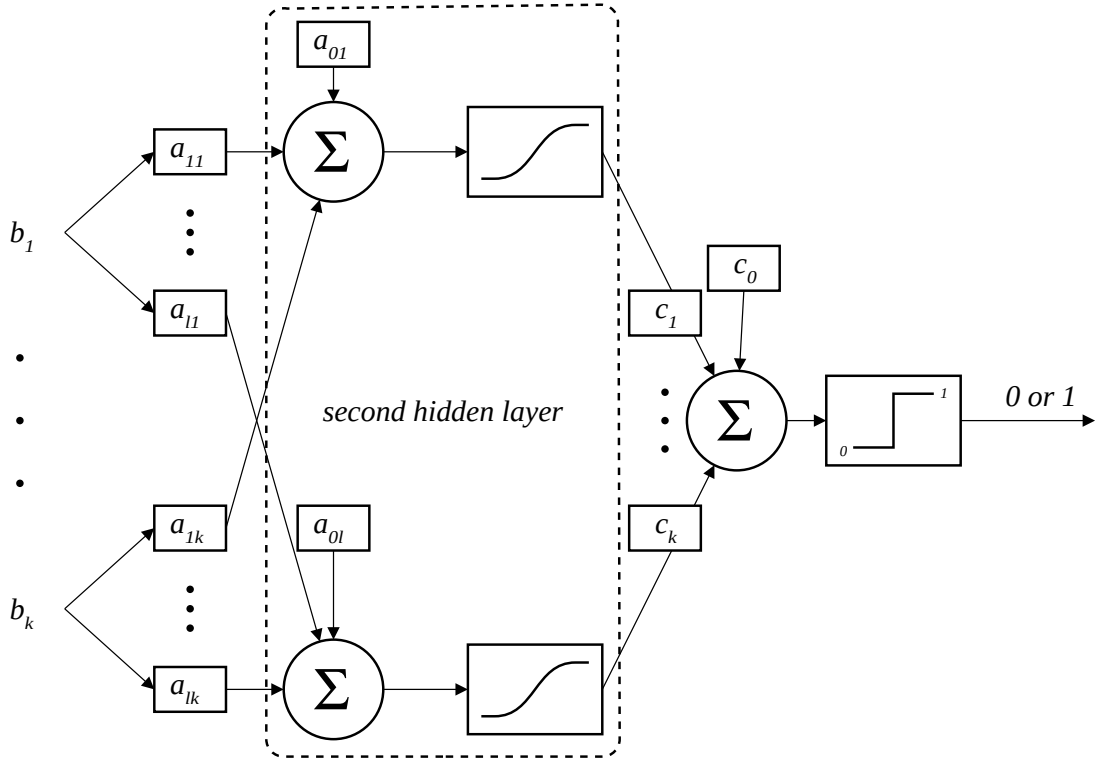
Arrangement Classifier

The decision regions for an arrangement classifier are *convex polytopes*, which are determined by intersections of half-spaces. The classifier is therefore *piecewise linear*.



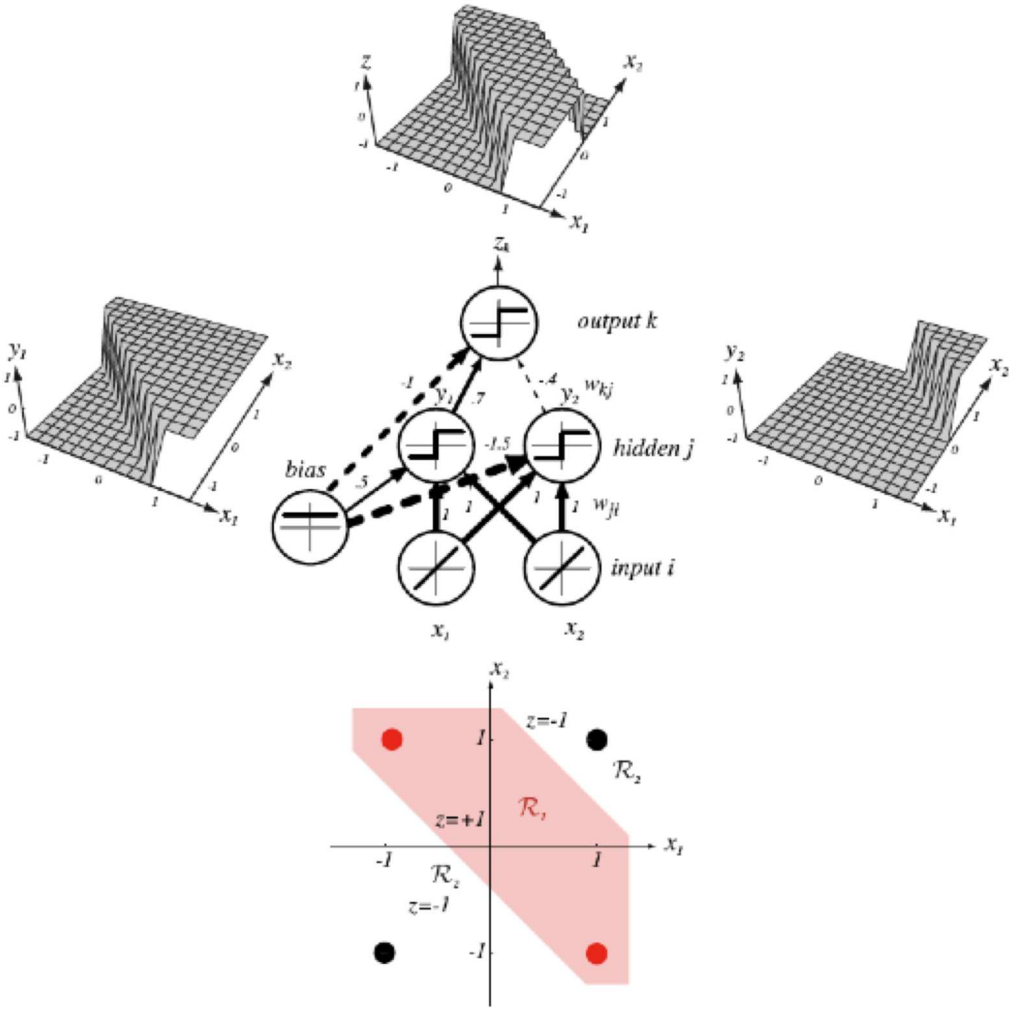
Arrangement Classifier - II

The decision regions in the arrangement classifier are determined by the first hidden layer of k threshold sigmoids. Each of the regions corresponds to a binary vector $b = [b_1, \ldots, b_k]$. A second (or all subsequent) hidden layers can only alter the class attributed to each of these regions.



XOR Problem

The neural network solution to the XOR problem is an arrangement classifier.



How to Pick the Parameters?

- Minimize the empirical error:

$$J(w) = \frac{1}{n} \sum_{i=1}^{n} |Y_i - \psi_n(X_i)|$$

- Minimize the *mean absolute error* (where $t_i = \pm 1$):

$$J(w) = \sum_{i=1}^{n} |t_i - \zeta(X_i)|$$

- Minimize the *mean square error*:

$$J(w) = \frac{1}{2} \sum_{i=1}^{n} [t_i - \zeta(X_i)]^2$$

(this leads to the *Backpropagation Algorithm*.)

Network Training

By using "augmented" vectors to represent the bias coefficients, we can write the output of a two-layer network with k hidden nodes as:

$$\zeta(x) = \sum_{i=0}^{k} c_i \xi_i(x)$$

where

$$\xi_i(x) \;=\; \sigma(\phi_i(x)) \;=\; \text{output of hidden node } i$$

and

$$\phi_i(x) \;=\; \sum_{j=0}^{d} a_{ij} x_j \;=\; \text{"activation" of hidden node } i$$

Network Training - II

The output can thus be written as

$$\zeta(x) = \sum_{i=0}^{k} c_i \sigma \left(\sum_{j=0}^{d} a_{ij} x_j \right)$$

The vector of parameters for this equation, also known as the *vector of weights* w, is given by:

$$w = \underbrace{(c_0, \ldots, c_k, a_{10}, \ldots, a_{1d}, \ldots, a_{k1}, \ldots, a_{kd})^T}_{(k+1)+k(d+1)=1+k(d+2) \text{ parameters}}$$

As k and d increase, the number of weights is roughly equal to $k \times d$.

Network Training - III

Given a training pattern (x, y), consider the mean-square error criterion

$$J(w) = \frac{1}{2}[t - \zeta(x)]^2$$

where the *target* t is given by:

$$t = \begin{cases} 1, & y = 1 \\ -1, & y = 0 \end{cases}$$

The total error over the entire training set is:

$$J_T(w) = \sum_{i=1}^{n} J_i(w) = \sum_{i=1}^{n} [t_i - \zeta(x_i)]^2$$

Network Training - IV

Our problem is to find the vector of weights w that minimizes the error criterion.

For this we need to choose a non-linear optimization technique. A few examples that are common in least-square problems (such as the current problem):

- Gauss-Newton Algorithm
- Levenberg-Marquardt Algorithm
- Gradient Descent ("Backpropagation Algorithm")

It should be noted that all these procedures are iterative.

Backpropagation Algorithm

The basic iteration step in gradient descent is given by:

$$\Delta w = -\ell \, \nabla J$$

that is, the vector of weights w is updated in the direction that decreases the error criterion, according to the step length ℓ.

By writing the above in component form, we get the updates for each weight separately:

$$\Delta w_i = -\ell \, \frac{\partial J}{\partial w_i}$$

The backpropagation algorithm consists of applying the chain rule to compute these partial derivatives.

Backpropagation Algorithm - II

The output of our two-layer network with k hidden nodes is:

$$\zeta(x) = \sum_{i=0}^{k} c_i \xi_i(x) = \sum_{i=0}^{k} c_i \sigma(\phi_i(x)) = \sum_{i=0}^{k} c_i \sigma\left(\sum_{j=0}^{d} a_{ij} x_j\right)$$

For the *hidden-to-output* weights c_i:

$$\frac{\partial J}{\partial c_i} = \frac{\partial J}{\partial \zeta} \frac{\partial \zeta}{\partial c_i} = -[t - \zeta(x)]\, \xi_i(x)$$

so that $\Delta c_i = \ell\, \delta^o \xi_i(x)$, where

$$\delta^o \equiv -\frac{\partial J}{\partial \zeta} = t - \zeta(x)$$

Backpropagation Algorithm - III

For the *input-to-hidden* weights a_{ij}:

$$\frac{\partial J}{\partial a_{ij}} = \frac{\partial J}{\partial \phi_i} \frac{\partial \phi_i}{\partial a_{ij}} = \frac{\partial J}{\partial \phi_i} x_j$$

Now,

$$\frac{\partial J}{\partial \phi_i} = \frac{\partial J}{\partial \zeta} \frac{\partial \zeta}{\partial \xi_i} \frac{\partial \xi_i}{\partial \phi_i} = -\delta^o \, c_i \, \sigma'(\phi_i(x))$$

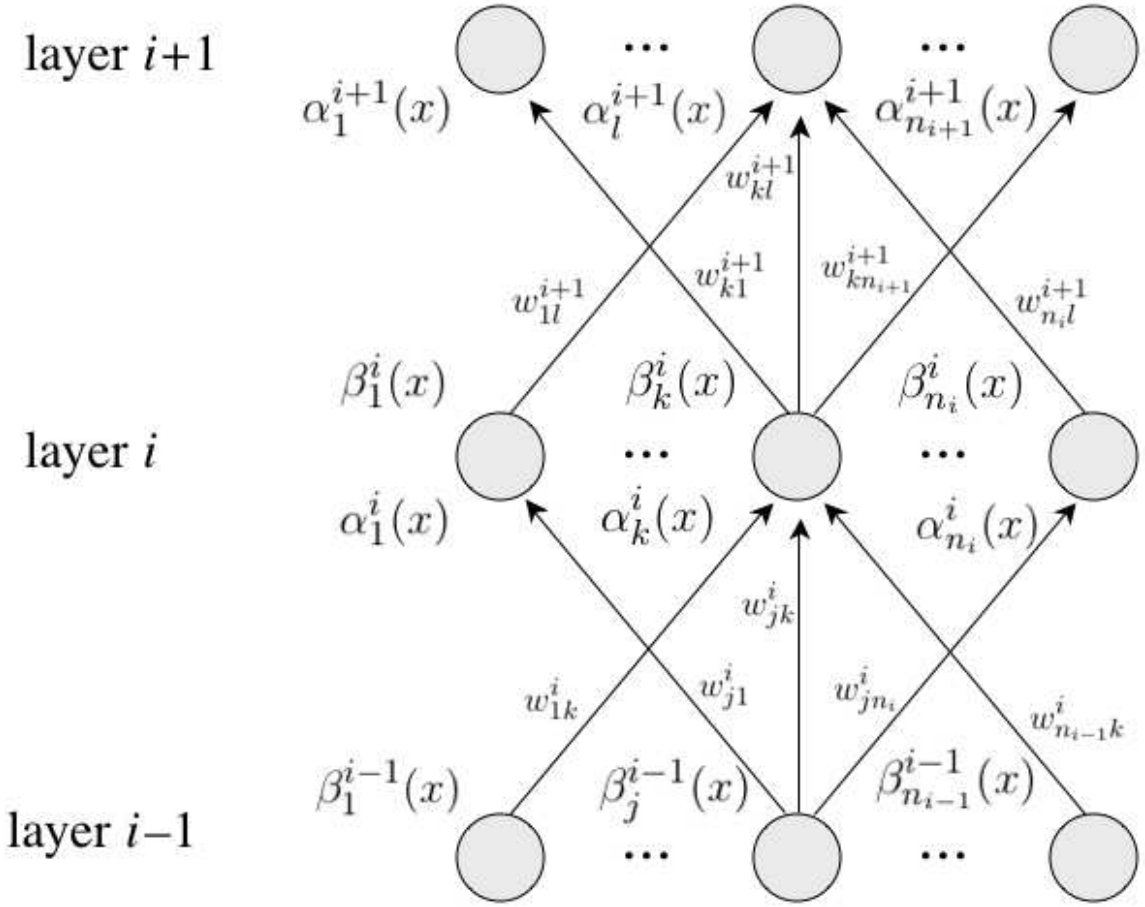
so that $\Delta a_{ij} = \ell \, \delta_i^H x_j$, where

$$\delta_i^H \equiv -\frac{\partial J}{\partial \phi_i} = \delta^o \, c_i \, \sigma'(\phi_i(x))$$

This is the backpropagation equation in our case, which relates the δ_i^H to δ^o, through the corresponding weight c_i.

General Backpropagation

In general, let $\alpha_j^i(x)$ and $\beta_j^i(x)$ be the activation and output for hidden node j in layer i of the network.



General Backpropagation - II

We have that

$$\beta_k^i(x) = \sigma(\alpha_k^i(x)) \quad \text{and} \quad \alpha_k^i(x) = \sum_{j=0}^{n_{i-1}} w_{jk}^i \beta_j^{i-1}(x)$$

Now,

$$\frac{\partial J}{\partial w_{jk}^i} = \frac{\partial J}{\partial \alpha_k^i} \frac{\partial \alpha_k^i}{\partial w_{jk}^i} = -\delta_k^i \, \beta_j^{i-1}(x)$$

where

$$\delta_k^i \equiv -\frac{\partial J}{\partial \alpha_k^i}$$

so that the update rule for weight w_{jk}^i is

$$\Delta w_{jk}^i = -\ell \frac{\partial J}{\partial w_{jk}^i} = \ell \, \delta_k^i \, \beta_j^{i-1}(x)$$

General Backpropagation - III

To determine δ_k^i one uses the chain rule:

$$\delta_k^i = -\frac{\partial J}{\partial \alpha_k^i} = -\sum_{l=1}^{n_{i+1}} \frac{\partial J}{\partial \alpha_l^{i+1}} \frac{\partial \alpha_l^{i+1}}{\partial \alpha_k^i} = \sum_{l=1}^{n_{i+1}} \delta_l^{i+1} \frac{\partial \alpha_l^{i+1}}{\partial \alpha_k^i}$$

while

$$\frac{\partial \alpha_l^{i+1}}{\partial \alpha_k^i} = \frac{\partial \alpha_l^{i+1}}{\partial \beta_k^i} \frac{\partial \beta_k^i}{\partial \alpha_k^i} = w_{kl}^{i+1} \sigma'(\alpha_k^i(x))$$

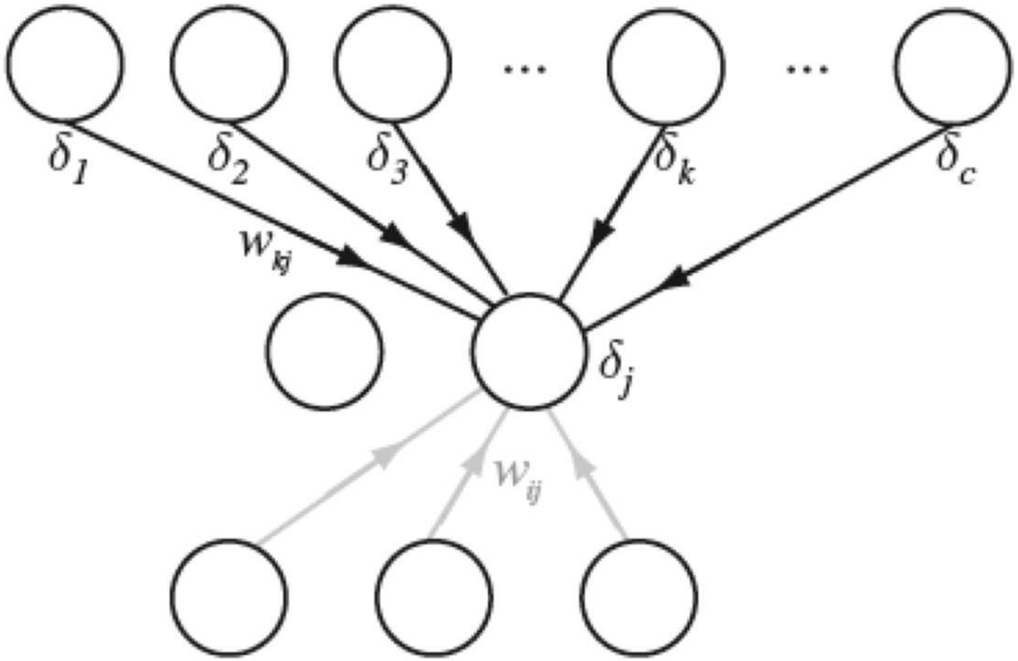
This results in the backpropagation equation:

$$\delta_k^i = \sigma'(\alpha_k^i(x)) \sum_{l=1}^{n_{i+1}} w_{kl}^{i+1} \delta_l^{i+1}$$

Hence, δ_k^i is determined by the deltas δ_l^{i+1} of the *next* layer.

General Backpropagation - IV

The deltas at one hidden layer are propagated backwards from the deltas in the next layer. Hence the term "backpropagation."



Backpropagation Training

- Initalize the weight vector with random numbers (not all zeros!).
- Present one training pattern at a time and update each weight according to the equation given previously.
- Stop when there is no significant change in the error criterion function.

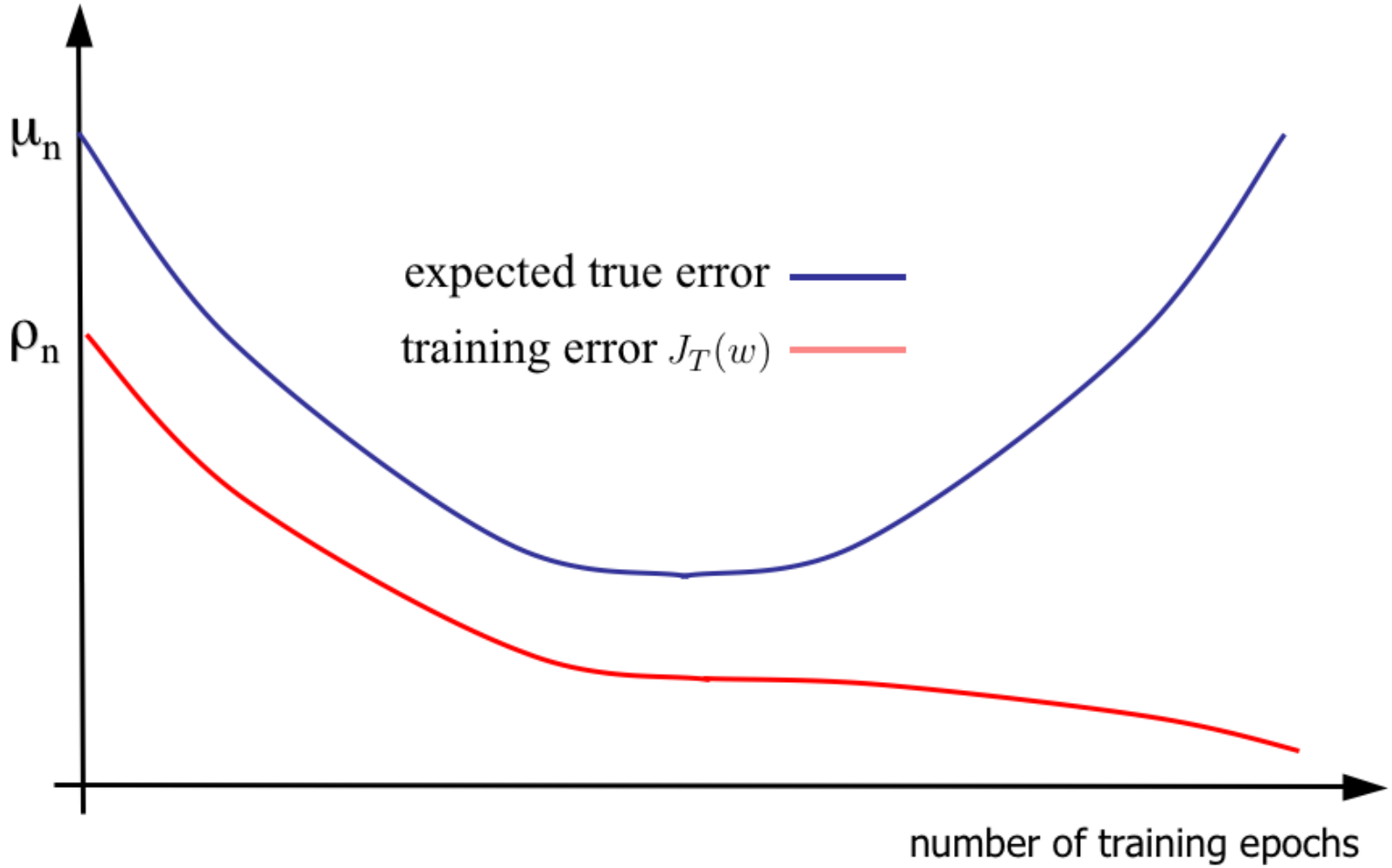
Notes:

1. A single presentation of the entire training set is called an *epoch*. The amount of training is measured by the number of epochs.

2. Each update is guaranteed to lower the individual error $J_i(w)$ for the pattern, but it could increase the total error $J_T(w) = \sum_{i=1}^{n} J_i(w)$. But, in the long run, $J_T(w)$ decreases.

Overfitting in NN Training

Even though $J_T(w)$ decreases in the long run, this is still the error on the training data, so there is danger of overfitting.



Consistency Result

This works for threshold sigmoids and empirical error minimization.

(DGL THM 30.7)
Let Ψ_n be the classification rule that uses minimization of the empirical error to design a neural network with k hidden nodes. If $k \to \infty$ such that $k \log n/n \to 0$ as $n \to \infty$, then Ψ_n is strongly universally consistent.

Drawback: It is not known how to do empirical error minimization efficiently.

Better Consistency Result

This works for arbitrary sigmoids and absolute error minimization, but the output weights are constrained.

(DGL THM 30.9)
Let Ψ_n be the classification rule that uses minimization of the absolute error to design a neural network with k_n hidden nodes with the constraint that the output weights satisfy

$$\sum_{i=0}^{k_n} |c_i| \leq \beta_n$$

If $k_n \to \infty$ and $\beta_n \to \infty$ and $\frac{k_n \beta_n^2 \log(k_n \beta_n)}{n} \to 0$ as $n \to \infty$, then Ψ_n is universally consistent. Note: this means that the number of hidden nodes and the bound on the output nodes must increase to infinity, but slowly (slower than n).