ANN

- Computational algorithm inspired by the parallelism of human brian
- Tries to mimic the cognitive ability of the human brian to identify patterns and to make decisions and forecasts based on past experience

How NN works?

- Non-linear models that can be trained to map the past and the future values of a time series
- extract hidden structure and relationship governing the data
- powerful pattern recognition capabilities

NN in the context of statistical methods

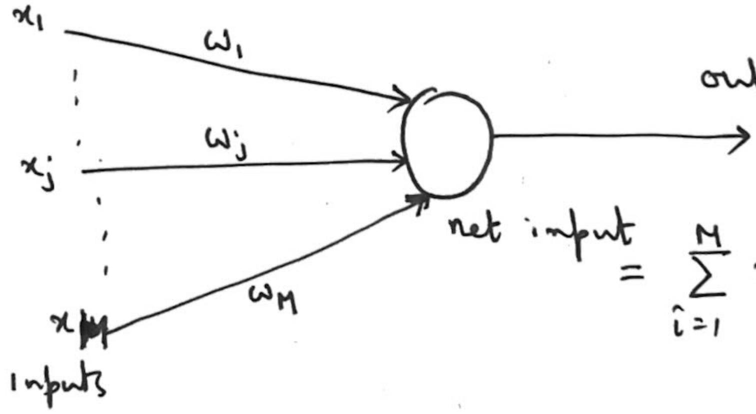
- Multivariate, non-linear, non-parametric inference technique
- completely data-driven and assumption free
- ability to model non-linear dynamics.

Some application areas

Stock market prediction, foreign exchange trading, bankruptcy prediction, volatility prediction, economic indicator forecasting credit rating, prediction of international airline passenger traffic, nowcasting in aviation industry, Ozone level prediction

Human brain: Hundred billion neurons - interconnected structure

Basic computational unit of ANN - a neuron - an artificial neuron



output = f(net input)
$= f\left(\sum_{i=1}^{N} w_i x_i\right)$.

net input $= \sum_{i=1}^{M} x_i w_i$

x_j : input value from the j^{th} input node

w_j : weight on the connection from the j^{th} input node to the PU.

PU - collects the inputs from all the inputs to get the net input value

$$\sum_{i=1}^{M} w_i x_i$$; w_i denoting the relative importance of i^{th} input

PU processes the net input value and generates the output

$$\text{Out} = f\left(\sum_{i=1}^{M} w_i x_i\right)$$

↑ transfer function / squashing f^n.

The simplest NN

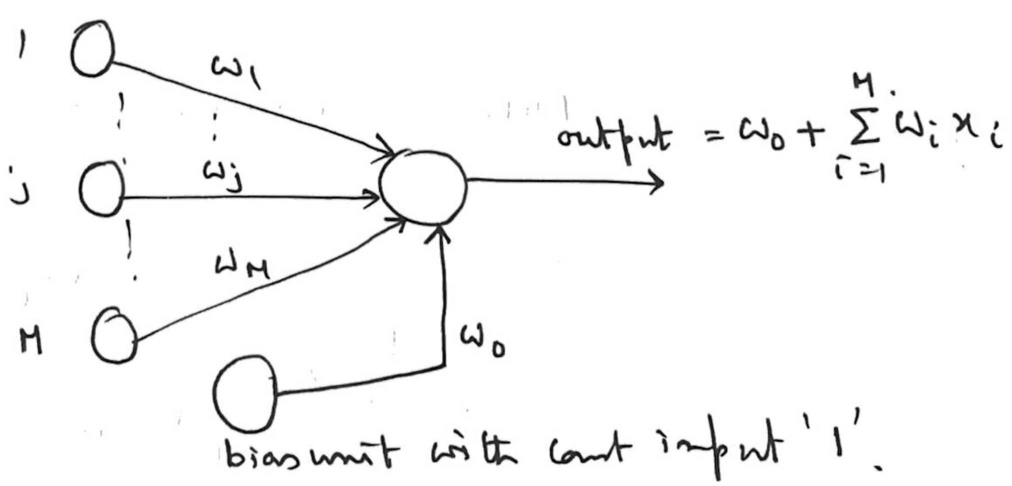
A single PU network.

MLR setup $\mathcal{L} = \{(\underset{\sim}{x}_1, y_1), \ldots (\underset{\sim}{x}_n, y_n)\}$.

Consider a transfer f^n as identity

$$\text{output} = f\left(\sum_{i=1}^{M} w_i x_i\right) = \sum_{i=1}^{M} w_i x_i$$

MLR model without a constant term



bias unit with const input '1'.

Unknown parameters : weight connections $(w_1, \ldots w_n)$ or $(w_0, w_1, \ldots w_n)$

optimum values of $(w_0, w_1, \ldots w_M)$ can be obtained by minimizing residual SSq.

<u>Note</u>: Injecting non-linearity into the model –

Consider a non-linear transfer f^n.

e.g (i) A sigmoidal transfer f^n.

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

(ii) A tanh transfer f^n

$$f(x) = \tanh(x).$$

For (i),
$$\text{output} = \frac{1}{1+e^{-(w_0 + \sum_{i=1}^{M} w_i x_i)}} = g(\underset{\sim}{x})$$

A non-linear model of response in terms of the values of the feature vector components

Let $(\underset{\sim}{x}_p, y_p)$ denote the p^{th} pattern in L.

Network output: $O_p = f\left(w_0 + \sum_{j=1}^{M} x_{pj} w_j\right) \Big/ . f\left(\sum_{j=1}^{M} x_{pj} w_j\right)$.

Actual output: y_p :

Error for the p^{th} pattern : $y_p - O_p$

Error square : $\mathcal{E}_p = \frac{1}{2}(y_p - O_p)^2 \leftarrow$ instantaneous error for p^{th} pattern

Total error over L

$$\mathcal{E} = \frac{1}{2}\sum_{p=1}^{n} \mathcal{E}_p = \frac{1}{2}\sum_{p=1}^{n}(y_p - O_p)^2.$$

Training of the network : determination of optimum weight vector.

Modes of Learning : Instanteneous mode or single pattern adaptation mode

or Batch mode or multiple pattern adaptation mode.

Method for obtaining the weight updation : Gradient descent algorithm

<u>Instanteneous mode</u> !

$$\mathcal{E}_p = \frac{1}{2}(y_p - o_p)^2.$$

$$\mathcal{E}_p = \frac{1}{2}\left(y_p - f\left(\sum_{j=1}^{M} \omega_j x_{pj}\right)\right)^2$$

Gradient of the error $\mathcal{E}_p$ w.r.t. the weights

$$\frac{\partial \mathcal{E}_p}{\partial \omega_j} = -(y_p - o_p) \cdot \frac{\partial f(\Sigma \omega_j x_{pj})}{\partial \omega_j}$$

$$= -(y_p - o_p) \frac{\partial f(\Sigma \omega_j x_{pj})}{\partial(\Sigma \omega_j x_{pj})} \cdot \frac{\partial(\Sigma \omega_j x_{pj})}{\partial \omega_j}$$

$$= -(y_p - o_p)\, f'(\Sigma \omega_j x_{pj})\, x_{pj}.$$

e.g. for sigmoidal transfer f^n.

$$f(\Sigma \omega_j x_{pj}) = \frac{1}{1 + e^{-\Sigma \omega_j x_{pj}}} = o_p.$$

$$\left[\; f(x) = \frac{1}{1+e^{-x}} \;;\; f'(x) = \frac{+e^{-x}}{(\quad)^2} = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right)\right]$$

$$f'(\Sigma \omega_j x_{pj}) = o_p(1 - o_p).$$

$$\Rightarrow \frac{\partial \mathcal{E}_p}{\partial \omega_j} = -(y_p - o_p)\, \cancel{f}\, o_p(1 - o_p)\, x_{pj}$$

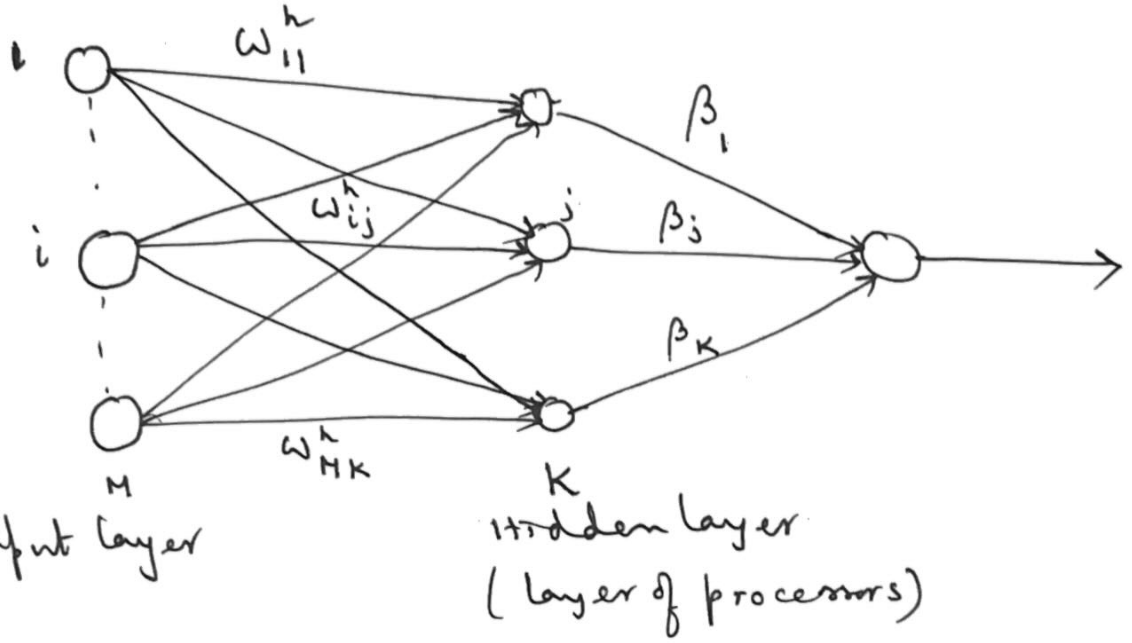
Gradient descent algorithm weight updation

$$\omega_j^{(k+1)} = \omega_j^{(k)} + \eta \left(- \frac{\partial \mathcal{E}_p}{\partial \omega_j} \right) \Bigg|_{\omega_j^{(k)}}$$

$$\omega_j^{(k+1)} = \omega_j^{(k)} + \eta \, (y_p - o_p) \, o_p (1 - o_p) \, x_{pj}$$

Present the patterns at random and use the above equation for weight updation untill stopping criterion is reached.

Multi-Layer Perceptron (MLP)



M input, K hidden layer neurons and one output

M - K - 1 MLP.

p th pattern $\rightarrow$ $(\underset{\sim}{x}_p, y_p)$

$$net_{pj}^h = \sum_{i=1}^{M} \omega_{ij}^h \, x_{pi} \quad \left(\text{may have a bias unit} \right).$$

net input value at j th neuron of hidden layer

output from the j th neuron

$$f(net_{pj}^h) = i_{pj}$$

i_{pj} : j^{th} input to the output unit

net input value at output

$$\sum_{j=1}^{K} \beta_j \, i_{pj} = net_p^o$$

Final network output

$$O_p = f(net_p^o)$$ (This 'f' can be diff from the f used at hidden layer neurons).

i.e.

$$\begin{aligned} O_p &= f(net_p^o) \\ &= f\left(\sum_{j=1}^{K} \beta_j \, i_{pj}\right) \\ &= f\left(\sum_{j=1}^{K} \beta_j \, f\left(\sum_{i=1}^{M} w_{ij}^h x_{pi}\right)\right). \\ \text{or } &= g\left(\sum_{j=1}^{K} \beta_j \, f\left(\sum_{i=1}^{M} w_{ij}^h x_{pi}\right)\right). \end{aligned}$$

Note: We can have different 'f' at different hidden layer neurons, in that case

$$O_p = g\left(\sum_{j=1}^{K} \beta_j \, f_j\left(\sum_{i=1}^{M} w_{ij}^h x_{pi}\right)\right).$$

Note: Suppose we consider sigmoid transfer f^n at all neurons (hidden and output) then

$$\begin{aligned} O_p &= f\left(\sum_{j=1}^{K} \beta_j \left\{1 + \exp\left(-\sum_{i=1}^{M} w_{ij}^h x_{pi}\right)\right\}^{-1}\right) \\ &= \left\{1 + \exp\left(-\sum_{j=1}^{K} \beta_j \left[1 + \exp\left(-\sum_{i=1}^{M} w_{ij}^h x_{pi}\right)\right]^{-1}\right)\right\}^{-1} \end{aligned}$$

i.e. $O_p = f^*(x_{p1}, \ldots, x_{pM}) \rightarrow$ a non-linear model.

Note: The network above is called a feed-forward network.

unknown parameters for an M-K-1 network : weight connections.

input layer $\rightarrow$ hidden layer : $M \times K$ weight matrix

$$\begin{pmatrix} \omega^h_{11} & \omega^h_{12} & \cdots & \omega^h_{1K} \\ - & - & - & - \\ \omega^h_{M1} & \omega^h_{M2} & \cdots & \omega^h_{MK} \end{pmatrix}.$$

hidden layer $\rightarrow$ output node : K-dimensional weight vector $(\beta_j, \ldots \beta_K)'$.

Total # of unknown weights : $(MK + K)$.

Learning of the network : determination of weights.

Either instantaneous mode OR batch mode. using Back-propagation algorithm (b-p of errors).

Consider for example learning on an instantaneous mode.

$$\mathcal{E}_p = \frac{1}{2}(y_p - o_p)^2 \leftarrow \text{instantaneous sq error for } p^{th} \text{ pattern}$$

$$\mathcal{E}_p = \frac{1}{2}\left(y_p - f^o(net^o_p)\right)^2 \quad \Big| \quad o_p = f^o\left(\sum_j \beta_j f^o_j(net^h_{pj})\right)$$

Gradient of $\mathcal{E}_p$ w.r.t. ~~output~~ hidden to output layer wts.

$$\frac{\partial \mathcal{E}_p}{\partial \beta^o_j} = -\cancel{\tfrac{1}{2}}(y_p - o_p)\frac{\partial f^o(net^o_p)}{\partial \beta^o_j}$$

$$= -\cancel{\tfrac{1}{2}}(y_p - o_p)\frac{\partial f^o(net^o_p)}{\partial net^o_p} \cdot \frac{\partial net^o_p}{\partial \beta^o_j}.$$

$net^o_p = \sum_{j=1}^{K} \beta^o_j i_{pj}$

$i_{pj} = f_j\left(\sum_{i=1}^{M} x_{pi} \omega^h_{ij}\right)$

$$= -\cancel{\tfrac{1}{2}}(y_p - o_p)\, f^{o\prime}(net^o_p)\, i_{pj}.$$

i.e. $$\frac{\partial \mathcal{E}_p}{\partial \beta^o_j} = -\cancel{\tfrac{1}{2}}(y_p - o_p)\, o_p(1 - o_p)\, i_{pj}$$

for a sigmoid transfer f^n at the output node

Sly gradient of $\mathcal{E}_p$ w.r.t. input layer → hidden layer wts

$$\frac{\partial \mathcal{E}_p}{\partial w^h_{ij}} = \frac{\partial}{\partial w^h_{ij}} \frac{1}{2}\left(y_p - f^o\left(\sum_{j=1}^{K} \beta_j f_j\left(\sum_{i=1}^{M} w^h_{ij} x_{pi}\right)\right)\right)^2$$

$$o_p = f^o(net^o_p)$$

$$net^o_p = \sum_{j=1}^{K} \beta^o_j i_{pj}$$

$$i_{pj} = f_j\left(\sum_{i=1}^{M} w^h_{ij} x_{pi}\right)$$

$$i_{pj} = f_j(net^h_{pj})$$

$$net^h_{pj} = \sum_{i=1}^{M} w^h_{ij} x_{pi}$$

$$= -(y_p - o_p)\frac{\partial f^o(net^o_p)}{\partial net^o_p} \cdot \frac{\partial net^o_p}{\partial i_{pj}} \cdot \frac{\partial i_{pj}}{\partial net^h_{pj}} \cdot \frac{\partial net^h_{pj}}{\partial w^h_{ij}}$$

$$= \underbrace{-(y_p - o_p)\, f^{o\prime}(net^o_p)}_{\delta^o_p}\, \beta^o_j\, f'_j(net^h_{pj})\, x_{pi}$$

Define, output layer delta as.

$$\delta^o_p = (y_p - o_p)\, f^{o\prime}(net^o_p)$$

and hidden layer delta as

$$\delta^h_{pj} = \delta^o_p\, \beta^o_j\, f'_j(net^h_{pj}).$$

Weight updation eq^{n}s

output layer weights

$$\beta^o_j(k+1) = \beta^o_j(k) + \eta\, \delta^o_p\, i_{pj}\Big|_{\underset{\sim}{\beta}^{(k)},\, W^{(k)}}$$

hidden layer weights

$$w^h_{ij}(k+1) = w^h_{ij}(k) + \eta\, \delta^h_{pj}\, x_{pi}\Big|_{\beta^{(k+1)},\, W^{(k)}}$$

η: learning rate parameter

Iteration continues till stopping criterion is reached

Either low error below threshold OR change in wt vector is lower than a threshold.

· Note: Information flows from input level to output level → error calculated on output layer ⟶ propagated back for adjustment of hidden to output layer wts ⟶ with updated hidden to output layer wts error propagates backwards for weight updation of input to hidden layer weights.

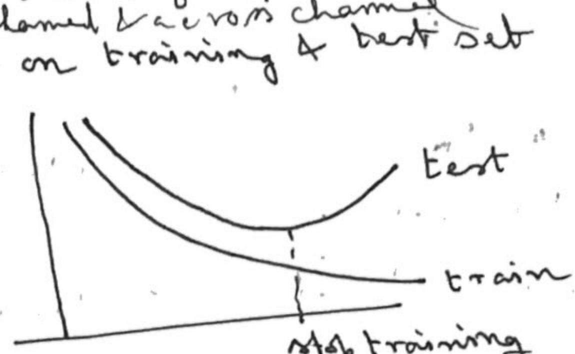
→ Back-propagation of error.

Some important points: (after the discussion on Variants)

(i) Data splitting - training set and test set

(ii) Preprocessing - normalization of inputs and output $\rightarrow \dfrac{* - \bar{x}_i}{Sd_{x_i}}$. $\rightarrow \dfrac{* - min}{max - min}$
Along channel & across channel

(iii) Overfitting - • MSE on training & test set



• weight decay - add a penalty term for large weights

$$R(\theta) + \lambda J(\theta)$$

$$R(\theta) = \sum_{p=1}^{n} (y_p - O_p)^2 \qquad \theta: \text{set of weights } (MK + K)$$

$$J(\theta) = \sum_{i,j} w_{ij}^{h\,2} + \sum_{K} \beta_K^{o\,2}$$

λ: tuning parameter, $\lambda \geq 0$
larger λ tend to shrink weights towards 0
(approach similar to Ridge regression

· weight elimination

$$J(\theta) = \sum_{i,j} \frac{w_{ij}^{h\,2}}{1 + w_{ij}^{h\,2}} + \sum_{K} \frac{\beta_K^{o\,2}}{1 + \beta_K^{o\,2}} .$$

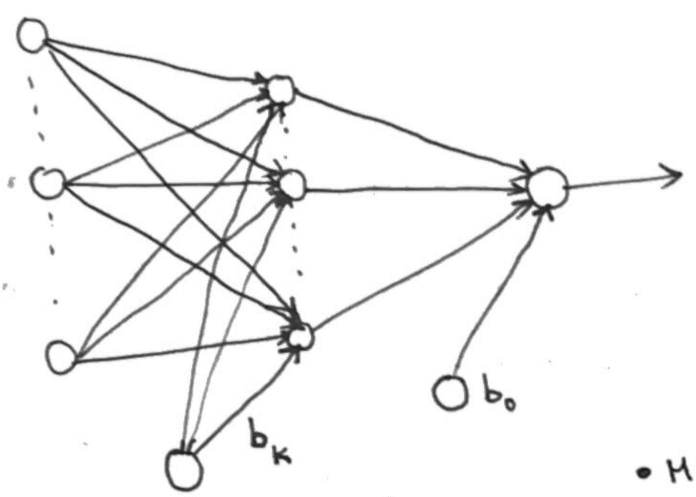
(iv) Number of hidden layers & units

One approach is to resort to experimentation

Systematic approach < Simulated annealing / Genetic algorithm

Variants of the feedforward MLP

• MLP with bias units

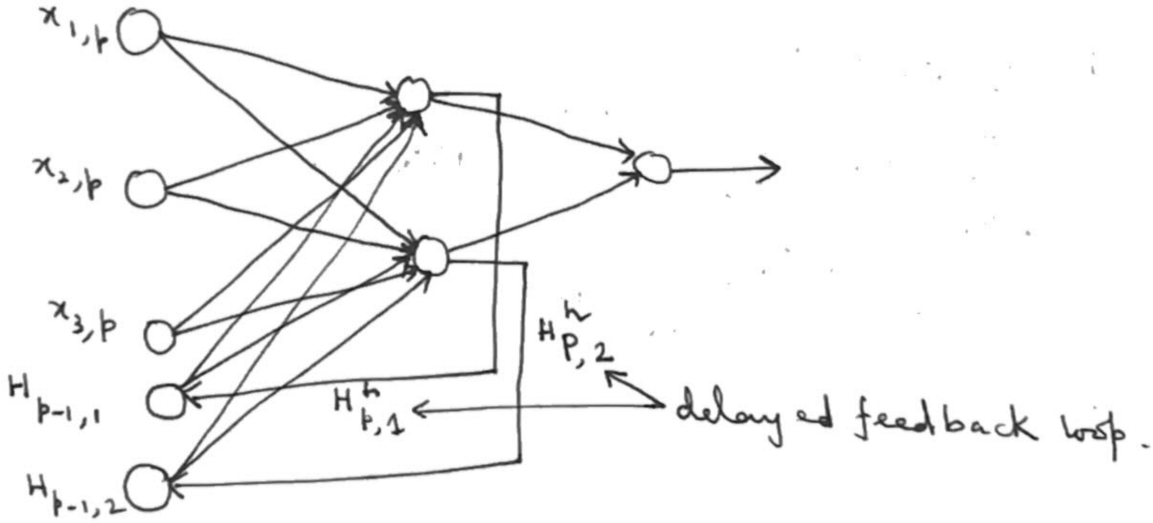


Value of bias 1

• Multiple output MLP.

• ~~MLP~~ Recurrent MLP (more appropriate for time series set up)

3-2-1

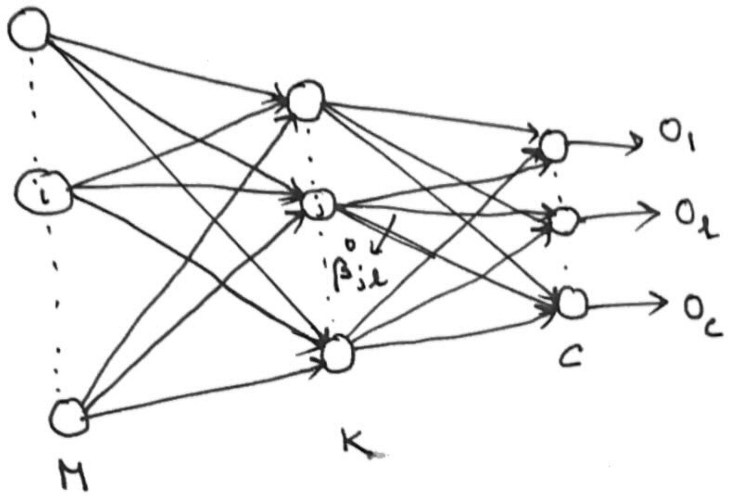


$$\hat{y}_p = \phi\left(\sum_{k=1}^{2} \beta_k^o \, \psi_k^h\left(\sum_{i=1}^{3} x_{pi}\, w_{ij}^h + \sum_{k=1}^{2} w_k^f \, \psi_k^h\left(\sum_{i=1}^{3} x_{p-1,i}\, w_{ij}^h\right)\right)\right)$$

feedback input reaching hidden units.

ANN classification model

c - class problem



$$\beta^o_{1l}\, i_{p1} + \dots \beta^o_{jl}\, i_{pj} + \dots$$

$$net^h_{pj} = \sum_{i=1}^{M} x_{pi}\, w^h_{ij}$$

$$i_{pj} = f^h_j\left(net^h_{pj}\right) = f^h_j\left(\sum_{i=1}^{M} x_{pi}\, w^h_{ij}\right)$$

At l^{th} output unit

inputs are $i_{p1}, i_{p2}, \dots i_{pK}$

Net input at l^{th} output $\displaystyle\sum_{j=1}^{K} \beta^o_{jl}\, i_{pj} = T_l \quad l = 1(1)C$

output of l^{th} output unit

$$O_l = \frac{e^{\cancel{i_{pl}}\, T_l}}{\sum_{j=1}^{C} \cancel{e^{i_{pj}}}\, e^{T_j}} \quad ; \quad l = 1(1)C$$

$$\sum_{l=1}^{C} O_l = 1 \quad ; \quad 0 \le O_l \le 1$$

Corresponding classifier is

$$G(\underset{\sim}{x}) = \underset{l}{\operatorname{argmax}}\, O_l$$

(*) Discuss how training set is to be constructed. This will specify what is y_{iK}.

Total error : $\displaystyle\sum_{i=1}^{n} \sum_{K=1}^{C} \left(y_{iK} - O_K\right)^2$

or deviance $\displaystyle -\sum_{i=1}^{n} \sum_{K=1}^{C} y_{iK} \log O_K(\underset{\sim}{x}_i)$

Steps for ANN model building

(1) Creation of Data records

(2) Splitting of data into training & test set

(3) Preprocessing of inputs and output

(4) Decide network architecture – # of inputs, # of hidden layers, # of neurons in hidden layer, transfer f^ns

(5) Training of the network

(6) Calculate error measure for training & test set