EE 583
PATTERN RECOGNITION

Feedforward Neural Networks

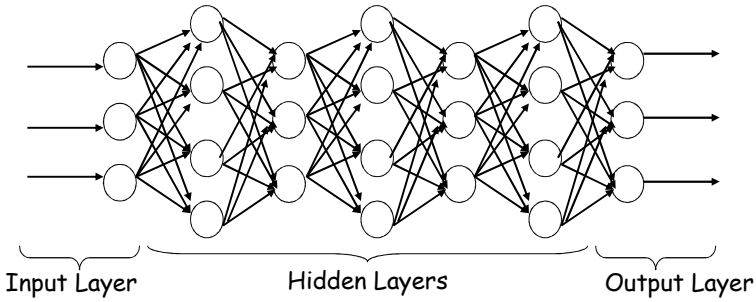
Multilayer Feedforward Network Structure

Training of Feedforward Networks

Practical Techniques for Improvement

Overall Feedforward Structure

- A typical multilayer feedforward network structure



- Training of such a network is necessary to determine the unknown weight values

Overall Feedforward Structure

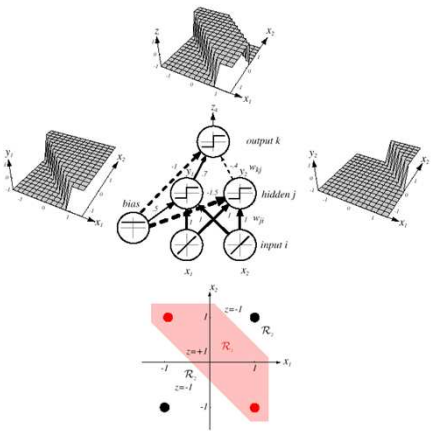
- Feedforward networks consist of two or more layers of processing units with implied directionality of the connections having no feedbacks

- The layers of feedforward networks :
  - *Input Layer* : Role is only to "hold" the values and distribute them to next layers
  - *Output Layer* : The layer at which the final state of the network is read.
  - *Hidden Layer(s)* : The layers between input and output layers, which are connected using weighted links to the higher levels.

    These internal layers should remap the inputs and results of previous layers to achieve a more seperable (classifiable) representation of data

Feedforward Operation

- Solving XOR problem using a 3-layer net



$$net_j = \sum_{i=1}^{d} x_i w_{ji} + \underbrace{w_{j0}}_{bias} = \sum_{i=0}^{d} x_i w_{ji} = \vec{w}_j{}^t \vec{x}$$

$$y_j = f(net_j) = \begin{cases} +1 & \text{if } net_j \geq 0 \\ -1 & \text{if } net_j < 0 \end{cases}$$

$$net_k = \sum_{j=1}^{n_H} y_j w_{kj} + w_{k0} = \sum_{j=0}^{n_H} y_j w_{kj} = \vec{w}_k{}^t \vec{y}$$

$$z_k = f(net_k)$$

For y_1 : $if \;\; x_1 + x_2 + 0.5 \geq 0 \Rightarrow y_1 = 1$

For y_2 : $if \;\; x_1 + x_2 - 1.5 \geq 0 \Rightarrow y_2 = 1$

For z_k : $if \;\; 0.7 y_1 - 0.4 y_2 - 1 \geq 0 \Rightarrow z_k = 1$

(i.e. $y_1 = 1 \;\&\; y_2 = -1$)

Hidden Layers

- There are two important questions which determine the structure of a neural network :
  - How many hidden layers are needed?
  - How many units should be in a hidden layer?

- There is a (non-constructive) existence proof by Kolmogorov:

  Given any continuous function, $f:I^d \rightarrow R^c$, $f(x)=y$ where I is the closed unit interval $[0,1]$, *f can be implemented exactly by a 3-layer neural network having*

  d processing elements in the input
  (2d+1) processing elements in the single hidden
  c processing elements in the output layer

$$\underbrace{z_k = \sum_{j=1}^{d} \psi_{j,k}(x_j)}_{\text{mapping function for the hidden layer}} \qquad \underbrace{g(\vec{x}) = \sum_{k=1}^{2d+1} h_k(z_k)}_{\text{mapping function for output}}$$

.... but no indication for how to construct $h_k(.)$ and $\psi_{j,k}(.)$ functions from a given $g(.)$function !

Training Feedforward Network (1/5)

- *Delta* and *Generalized Delta Rules* use gradient descent to achieve training
- The basic operation (*Backpropagation Algorithm*)
  1. Apply (training) stimulus to network
  2. Feedforward (propagate) input to determine outputs
  3. Compare outputs against the desired response
  4. Compute and propagate error measure backward through network while minimizing error at each stage via weight adjustments

- Terminology is as follows :

  $\underline{i}$: input pattern (dx1)
  $\underline{o}$: output pattern (cx1)
  $\underline{w}$: network weights
  $\underline{t}$: desired system output

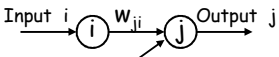
- Weight w_{ji} is defined FROM unit-i TO unit-j

Training Feedforward Network (2/5)

- Assume a 2-layer network with no hidden units
- Assume *n* input-output pairs for training

$$H = \left\{(\underline{i}^k, \underline{t}^k)\right\} \quad k = 1,\ldots,n \quad \text{where } j^{th} \text{ element is } i_j^p \in \underline{i}^p$$

- A product-based weight correction strategy :



$$\Delta^p w_{ji} = \varepsilon(t_j^p - o_j^p) i_i^p \quad (\varepsilon : \text{scaling parameter})$$

- Reasoning behind the above correction strategy :
  - if $i_i^p > 0$ then weight must increase (decrease) to make o_j^p reach the desired response t_j^p
  - if $i_i^p < 0$ then weight must work in the reverse way
  - if $i_i^p = 0$ then no correction (not desirable)
- The same training strategy can be applied after using the whole training set (*epoch*) at each weight update

$$\Delta w_{ji} = \sum_p \Delta^p w_{ji}$$

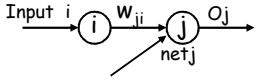
Training Feedforward Network (3/5)

- Lets formalize the previous weight correction strategy :

$$\text{Error vector} : \underline{e}^p \equiv \underline{t}^p - \underline{o}^p \quad \Rightarrow \quad E_p = \frac{1}{2}(\underline{e}^p)^T \underline{e}^p = \frac{1}{2}\left\|\underline{e}^p\right\|^2$$

- The relation between output error E_p and weight w_{ji} can be written as $\dfrac{\partial E_p}{\partial w_{ji}}$
- This relation can be written in the form below (via chain rule) :

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_j^p} \frac{\partial o_j^p}{\partial net_j^p} \frac{\partial net_j^p}{\partial w_{ji}}$$

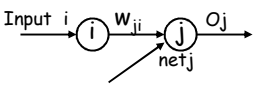


$$net_j = \sum_i w_{ji} i_i$$

$o_j = f_j(net_j)$ where f is a nondecreasing differentiable function

Training Feedforward Network (4/5)

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_j^p}\frac{\partial o_j^p}{\partial net_j^p}\frac{\partial net_j^p}{\partial w_{ji}}$$



- Each term in the equation above can be written as :

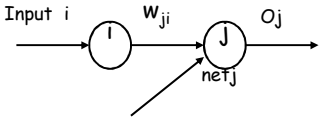
$$\frac{\partial E_p}{\partial o_j^p} = \frac{\partial\left(\frac{1}{2}\left(\underline{t}^p - \underline{o}^p\right)^T\left(\underline{t}^p - \underline{o}^p\right)\right)}{\partial o_j^p} = -\left(t_j^p - o_j^p\right) = -e_j^p$$

$$\frac{\partial o_j^p}{\partial net_j^p} = \frac{\partial f(net_j^p)}{\partial net_j^p} \quad if \quad f(x) = \frac{1}{1+e^{-x}} \Rightarrow \frac{\partial o_j^p}{\partial net_j^p} = \frac{e^{-net_j^p}}{(1+e^{-net_j^p})^2} = o_j^p(1-o_j^p)$$

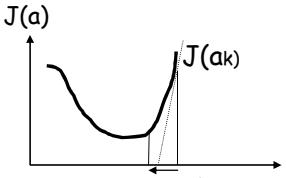
$$\frac{\partial net_j^p}{\partial w_{ji}} = \frac{\partial\left(\sum_i w_{ji} i_i^p\right)}{\partial w_{ji}} = i_i^p \quad \text{(for a multilayer net } i_i^p \text{ : output of previous layer)}$$

Training Feedforward Network (5/5)

$$\frac{\partial E_p}{\partial w_{ji}} = \underbrace{\frac{\partial E_p}{\partial o_j^p}\frac{\partial o_j^p}{\partial net_j^p}}_{\equiv -\delta_j^p}\frac{\partial net_j^p}{\partial w_{ji}}$$



- Delta Rule : In order to minimize E_p wrt w_{ji}, move (create an update term) in the direction of the negative gradient of E_p :



$$a_{k+1} = a_k - \underbrace{\rho_k}_{\text{step size}} \nabla J(a_k)$$

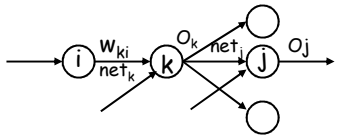
$$w_{ji}(n+1) = w_{ji}(n) + \Delta^p w_{ji}$$

$$\Delta^p w_{ji} = -\varepsilon\frac{\partial E^p}{\partial w_{ji}} = +\varepsilon\underbrace{\left(o_j^p(1-o_j^p)\right)\left(t_j^p - o_j^p\right)}_{\delta_j^p} i_i^p \quad \left(\varepsilon : \begin{matrix} learning \\ rate \end{matrix}\right)$$

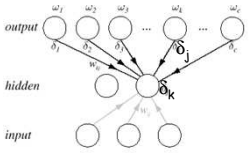
Generalized Delta Rule

$$\frac{\partial E_p}{\partial w_{ki}} = \underbrace{\frac{\partial E_p}{\partial o_k^p}\frac{\partial o_k^p}{\partial net_k^p}}_{-\delta_k^p}\frac{\partial net_k^p}{\partial w_{ki}}$$

$$\frac{\partial o_k^p}{\partial net_k^p} = o_k^p(1-o_k^p) \qquad \frac{\partial net_k^p}{\partial w_{ki}} = i_i^p$$



$$\frac{\partial E_p}{\partial o_k^p} = \sum_{j=1}^{c}\underbrace{\frac{\partial E_p}{\partial net_j^p}}_{-\delta_j^p}\frac{\partial net_j^p}{\partial o_k^p} = -\sum_{j=1}^{c}\delta_j^p w_{jk} \Rightarrow \delta_k^p = o_k^p(1-o_k^p)\sum_{j=1}^{c}\delta_j^p w_{jk}$$



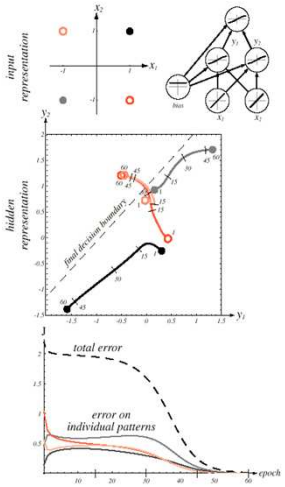
- Back Propagation Algorithm :
  1. Give training stimulus, obtain outputs
  2. Update weights of output layer using Delta Rule
  3. For hidden layers, use Generalized Delta Rule
  4. Stop if error is insignificant; else goto step 1

Training Protocols

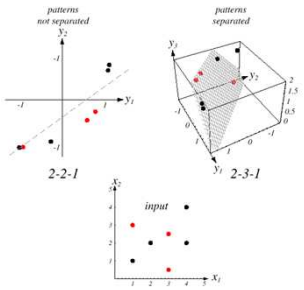
- There are three important training protocols for supervised training of a feedforward network
  - Stochastic training
    - Training patterns are chosen randomly from the training set
    - Networks weights are update after each pattern
  - Batch training
    - All patterns are presented to the network before learning
    - Update terms are summed up for all data and added after one "epoch" (a single presentation of all elements of training set)
  - On-line training
    - Each pattern is presented to the system one and only one time
    - No use of memory/storage for storing training data
- Stopping is usually achieved when the change in E_p between two epoch is less than a threshold δ

Backpropagation as Feature Mapping

- For XOR problem, the outputs of hidden layers become linearly separable after iterations ➔



- As another example, a sigmoidal network with topology 2-3-1 performs better to that of2-2-1

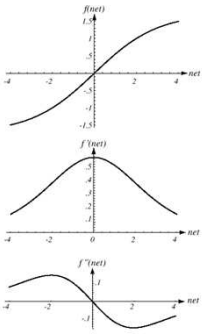


Practical Techniques for Improving Performance of Feedforward Nets

- During training of a neural net, the process can
  - be very slow (perhaps stuck in a local minimum)
  - becomes unstable
  - oscillating between solutions
- Some possible "rule-of-thumbs" to find a "good" solution
  - Choose a "good" activation function
  - Scale your input
  - Train with noise and manufactured data
  - Modify the number of hidden units and hidden layers
  - Check the initial weight values
  - Modify the learning rate
  - Use momentum or corresponding weight
  - Decay your weights
  - Add some classification hints to make training easier
  - Stop training if the system becomes memorizing (over-fitting)

Practical Techniques : Activation Function

- Backpropagation works on any activation function, if f is a continuous non-linear function with its derivative existing
  - partially continuous case requires extra constraint
- An important property of f :
  - saturation ➔ weights will be bounded
- Monotonic function ($f'(.)>0$ or $f'(.)<0$) is preferred but not essential
  - with non-monotonic functions, extra local minima may occur in the error surface)
- A function, satisfying all the above properties
  ➔ Sigmoid function:



$$f(net) = a\tanh(b\,net) = a\left[\frac{1-e^{-b\,net}}{1+e^{-b\,net}}\right] = \frac{2a}{1+e^{-b\,net}} - a$$

If $a = 1.716$ and $b = 2/3$

$\Rightarrow f'(0) \approx 1$, extrema of $f''(\cdot)$ at ± 2, linear range $-1 \le x \le +1$

METU EE 583 Lecture Notes by A.Aydin ALATAN © 2011

Practical Techniques : Scaling Input

- Different features, whose numerical values are <u>different on the order of magnitude</u> from each other, will create a bias during training
  - ➔ Weight adjustment will favor feature with larger numbers
- In order to avoid such problems, <u>normalize features</u> such that
  - the mean of the features for the training data is zero
  - the variance is adjusted a predetermined value
- Such an approach is only possible for the batch systems;
  - For on-line data, such a normalization may not be useful
- Output target values should be chosen to be ± 1, although the saturating value of the sigmoid is 1.716 (as input → ∞)

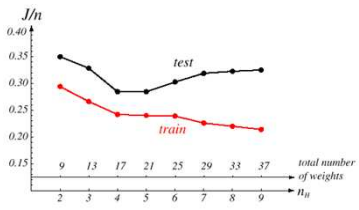
Practical Techniques : Artificial Train Data

- If the training set is small, one option is to generate some "virtual" training data by adding some Gaussian noise on top of the available data
  ➔ the training set increases

- If one has information about the sources of the variation among patterns, manufacturing training data might be preferable
  - e.g. rotating a pattern will generate manufactured data
  - a drawback is memory requirements and overall a slow training

Practical Techniques : Hidden Units

- The number of input and output units are determined by dimension of the input data and number of categories, respectively
- The number of hidden units, n_H , is related with the complexity of decision boundary
  - If data is well (linearly) separated → n_H might be small



For a 2-n_H-1 network, the cost function (per training data) :

- always decrease for training data

- may increase for test data after some value of n_H (memorization)

- Another approach is to adjust the complexity of the network
  ➔ start with large number of units, then prune weights

Practical Techniques : Hidden Layers

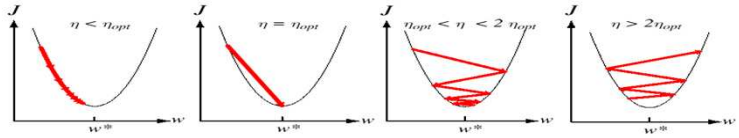
- The theorem by Kolmogorov indicates the sufficiency of 3 layers (a single hidden layer) for representing any function
- However, in practice for some cases, more than 3 layers may be required
- One possible requirement is due to learning of some invariants within a limited range of parameters at every layer
  - e.g. Character Recognition : a single hidden layer can learn characters which are translated for up to n pixels. Multiple layers may increase this limit of translation
- Networks with multiple hidden layers are found out to be trapped into local minima more often
- In case of absence of a problem specific reason, the first case to check should consist of a single hidden layer

Practical Techniques : Initialize Weights

- Never initialize weights to zero ! ➔ zero weight updates
- Initial weights should be selected such that they yield to "uniform learning"
  - All weights reach to equilibrium values at the same time
- Choose weights randomly from the same distribution
- Since the scaled data is zero mean (+ and – values), the weights should also change between $-w_m < w < +w_m$
  - If w_m is too small → stuck in linear region of activity fnct.
  - If w_m is too large → saturation before learning
- In order to obtain $-1 < net < +1$ at the input of a unit with d connections from input layers with scaled unit variance
  - ➔ $-1/\sqrt{d} < w < +1/\sqrt{d}$
- Similarly, for the output layer for a network with n_H units in the hidden layer, the initialization should be
  - ➔ $-1/\sqrt{n_H} < w < +1/\sqrt{n_H}$

Practical Techniques : Learning Rates

- In principle, if enough time is spent to let a network to reach its minimum error, parameter for learning rate, ε (or η) , can be selected as any small value, but in practice this is not the case



- Optimal learning rate is the one that leads to local error minimum in a single step
- If the criterion function can be approximated by a quadratic, the optimal parameter is obtained using 2nd order derivatives :

$$J(a) \approx J'(a) = J(a_k) - \nabla J^t (a - a_k) + \frac{1}{2}(a - a_k)^t \underbrace{D}_{\substack{2nd \\ deriv.}} (a - a_k) \quad \text{and let } a = a_{k+1} = a_k - \varepsilon_k \nabla J(a_k)$$

$$\min_{\varepsilon} J'(a_{k+1}) = \min_{\varepsilon} \left\{ J(a_k) - \varepsilon_k \|\nabla J\|^2 + \frac{1}{2} \varepsilon_k{}^2 \nabla J^t D \nabla J \right\} \Rightarrow \varepsilon_{optimal} = \frac{\|\nabla J\|^2}{\nabla J^t D \nabla J}$$

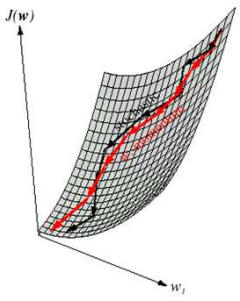
- In practice, ε = 0.1 is often enough as a first choice

Practical Techniques : Adding Momentum

- If there are too many weights, the error surface will consist of regions with small slope, in which weight update is slow
- In order to add momentum to weight update process at (n+1)th iteration, the correction is modified :

$$\Delta^p w_{ji}(n+1) = +\varepsilon\left(o_j^p (1 - o_j^p)\right)(t_j^p - o_j^p) i_i^p + \alpha \Delta^p w_{ji}(n)$$

- For positive alpha, the second term yields a correction that is somewhat larger (in the same direction) than the case without a momentum term
- Momentum term may
  - prevent oscillations
  - help escaping local minima (momentum built-up during entry into minima from one side may be enough to make it escape from the other side of local minima)

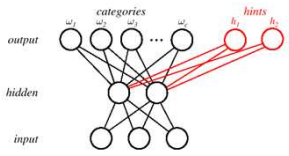


Practical Techniques : Weight Decay

- One method of simplifying a network and avoiding overfitting is to
  - begin from a network with too many weights,
  - "decay" every weight at each weight update by
    $$w^{new} = w^{old}\,(1-\beta)\ (0<\beta<1)$$
  - eliminate some of the weights with very small values,
- The weights which are really necessary will stay indefinitely
- It can be shown that such an approach is equivalent to a cost function to be minimized :
  $$J_{ef} = J(w) + \rho\, w^t w$$
- The system achieves a balance between pattern error and overall weight
- Although, there is no reason for such a method to *always* improve system performance, for most of the cases, this method yields improved results

Practical Techniques : Hints

- In some problems, it is possible to add some extra information during training to improve the classification performance
- Some properties of the input data can be used during training as a "hint" to the system, so that better classification can be achieved



Note that input to hidden layer weights will improve

- After training stage, the hint outputs are neglected in the classification stage
- A typical example is the problem of classification of c phonemes
  - ➔ add 2 hint classifying outputs for vowel and consonants
  - ➔ distinguish /b/ from /oo/ or /g/ from /ii/ better

Practical Techniques : Stopped Training

- Excessive training can lead to poor generalization as the system implements a complex decision boundary
- In such a case, the system is simply "tuned" to the given data
- For linear discriminant functions, there is no such problem, since the boundaries are always hyperplanes
- A simple method for avoiding this problem is to stop when the error on a separate validation set reaches a minimum