Unit-II

Artificial Neural Networks and Evaluation Hypothesis

- Introduction to ANN
- Neural Network Representations
- Appropriate Problems for Neural Network Learning
- Perceptron
- Multilayer Networks and the Backpropagation Algorithm
- Remarks on the Backpropagation Algorithm
- An Illustrative Example: Face Recognition
- Advanced Topics in Artificial Neural Networks

- Motivation for Evaluating Hypothesis
- Estimating Hypothesis Accuracy
- Basics of Sampling Theory
- A General Approach for Deriving Confidence Intervals
- Difference in Error of Two Hypothesis
- Comparing Learning Algorithms

Artificial Neural Networks-Introduction

- Neural Network Learning methods provide a robust approach to approximating:
  - Real valued
  - Discrete valued and
  - Vector valued target functions
- For certain types of problems like learning to interpret complex real-world sensor data artificial neural networks are the most effective learning methods currently known (using backpropagation).

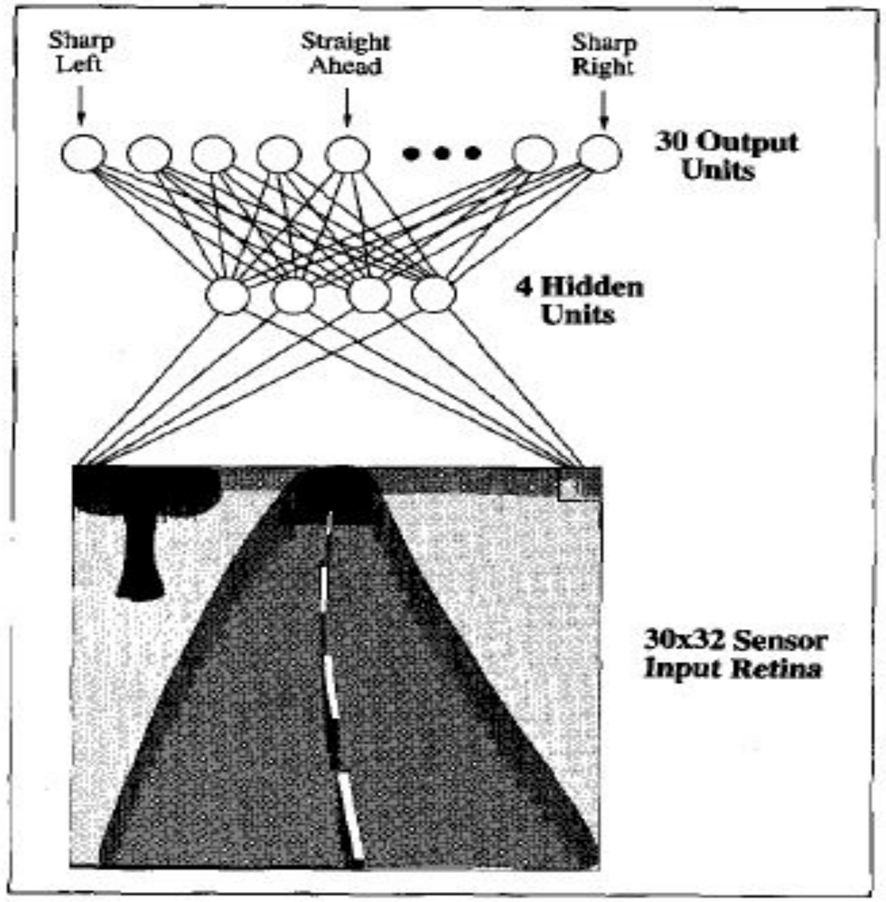
Introduction- Biological Motivation

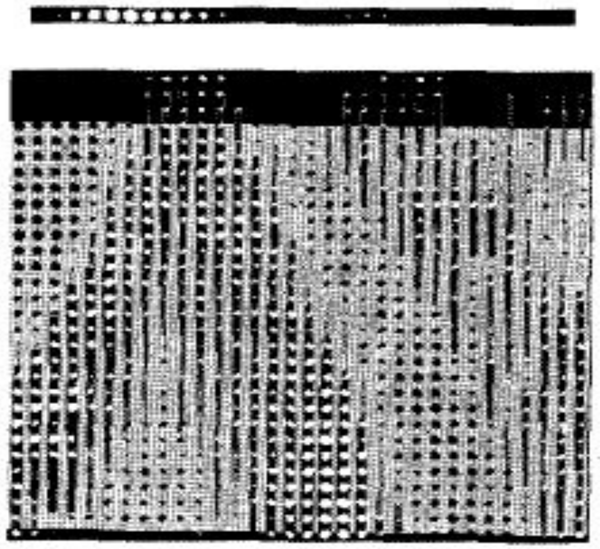
- The study of artificial neural networks (ANNs) has been inspired in part by the observation that biological learning systems are built of very complex webs of interconnected neurons.
- Artificial neural networks are built out of:
  - densely interconnected set of simple units
  - each unit takes a number of real-valued inputs
  - produces a single real-valued output.

Introduction- Biological Motivation

- The brain contains 10^{11} neurons each of which may have up to 10^{4} connections.
- Each neuron’s switching time is up to 10^{-3} seconds which is very slow in comparison to computers which have a switching time of 10^{-10} sec.
- At that slow switching time the brain is very fast at computationally intensive tasks like vision, speech recognition and storage retrieval.
- “Neural nets” or “connectionist models” are based on the assumption that computational architecture similar to the brain will duplicate its wonderful abilities.

Neural Network Representations-ALVINN System





Appropriate Problems for Neural Network Learning

- ANN learning is suitable for problems in which the training data corresponds to noisy data and complex sensor data from cameras and microphones.
- It is also applicable to problems where decision trees are used.
- The backpropagation algorithm is the most commonly used ANN learning technique.

Appropriate Problems for Neural Network Learning

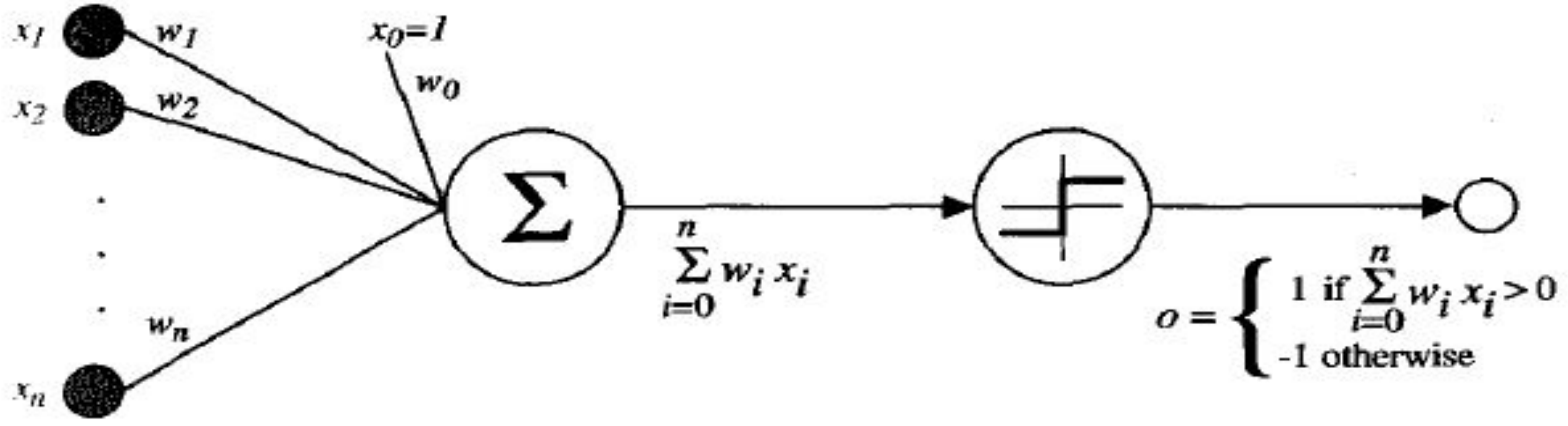
- ANN learning is appropriate for problems with the following characters:
    - Instances are represented by many attribute-value pairs.
    - The target function output value may be discrete, real or a vector of several real or discrete valued attributes.
    - The training examples may contain errors.
    - Long training times are acceptable.
    - Fast evaluation of the learned target function may be required.
    - The ability of humans to understand the learned target function is not important.

Perceptron

- Introduction
- Representational Power of Perceptron
- The Perceptron Training Rule
- Gradient descent and the Delta rule
- Remarks

Perceptron-Introduction

- One type of ANN system is based on a unit called a perceptron.
- A perceptron:
    - Takes a vector of real-valued inputs
    - Calculates a linear combination of these inputs
    - Outputs a 1 if the result is greater than some threshold and -1 otherwise.


x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

Perceptron-Introduction

- Given inputs x1,x2,……xn the output o(x1,x2,…..xn) computed by the perceptron i

$$o(x_1,\ldots,x_n) = \begin{cases} 1 \text{ if } w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

- Where each w_i is a real valued constant or weight that determines the contribution of input x_i to the perceptron output.
- The quantity -w0 is the threshold that the weighted combination of inputs w1x1+w2x2+…..+wnxn must surpass in order for the perceptron to give an output of 1.

Perceptron-Introduction

- We imagine an additional constant input x0=1 to write the above equation as:

$$\sum_{i=0}^{n} w_i x_i > 0$$

- The same equation can be written in the vector form as:

$$\vec{w} \cdot \vec{x} > 0$$

- The perceptron function can be written as:

$$o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$$

where

$$sgn(y) = \begin{cases} 1 & \text{if } y > 0 \\ -1 & \text{otherwise} \end{cases}$$

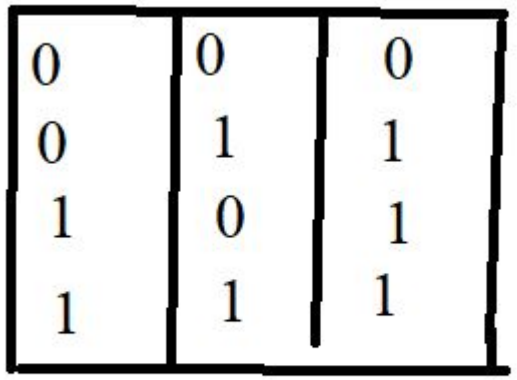
Representational Power of Perceptron

- The perceptron can be represented as a hyperplane decision surface in the n-dimensional space of instances.
- The perceptron outputs a 1 for instances lying on one side of the hyperplane and outputs a -1 for instances lying on the other side.
- The equation for this decision hyperplane is $\vec{w} \cdot \vec{x} = 0$
- The sets of positive and negative examples that can be separated by any hyperplane are called as linearly separable sets of examples.

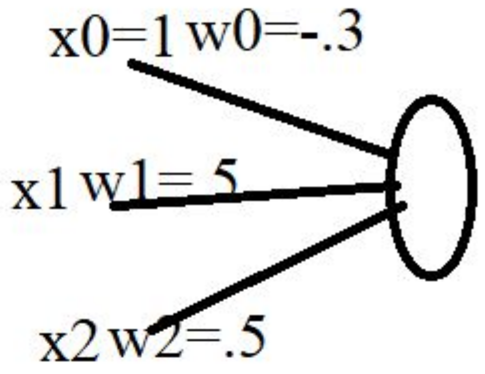
Representational Power of Perceptron

- A single perceptron can be used to represent many Boolean functions.
- If we assume Boolean values of 1 (true) and -1 (false), the perceptron can represent AND operation (w0=-.8 and w1=w2=.5).
- The perceptron can also represent OR operation (w0=-.3 and w1=w2=.5).
- AND and OR operations can be viewed as special cases of m-of-n functions:
  - functions where at least m of the n inputs to the perceptron must be true.
  - the OR function corresponds to *m* = 1 and the AND function to m = n

Example-OR function



0	0	0
0	1	1
1	0	1
1	1	1



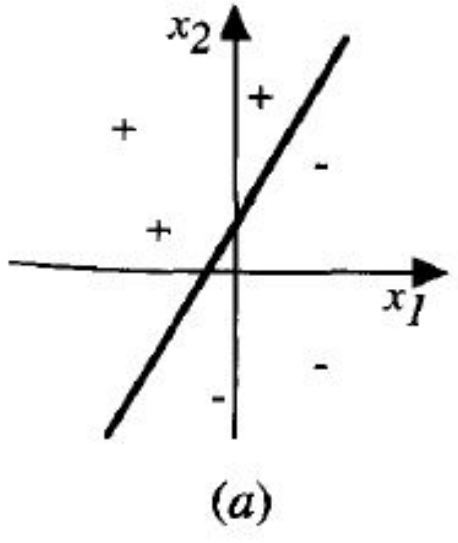
w0x0+w1x1+w2x2

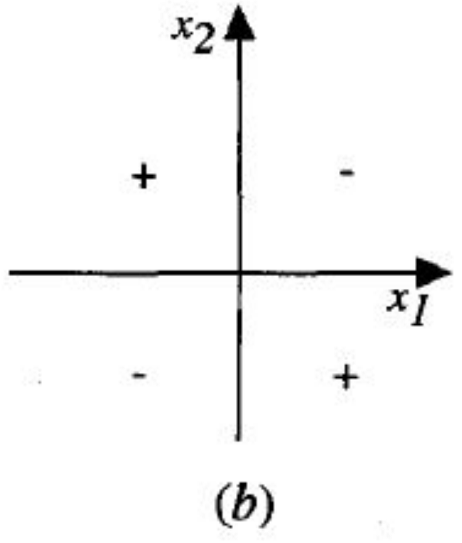
-.3*1+.5*0+.5*1=-.3+.5=.2>0 output =1

-.3*1+.5*0+.5*0=-.3<0 output =-1

Representational Power of Perceptron

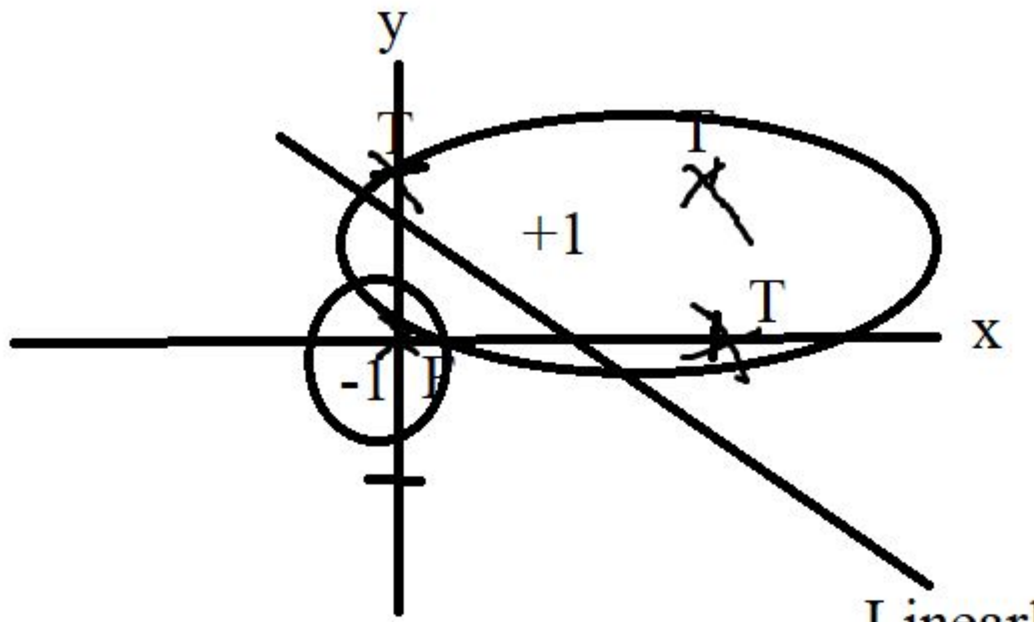
- Perceptron can represent Boolean functions like AND,OR, NAND and NOR.
- Perceptron cannot represent XOR function whose value is 1 iff x1!=x2.





Representational Power- Linearly Separable

OR

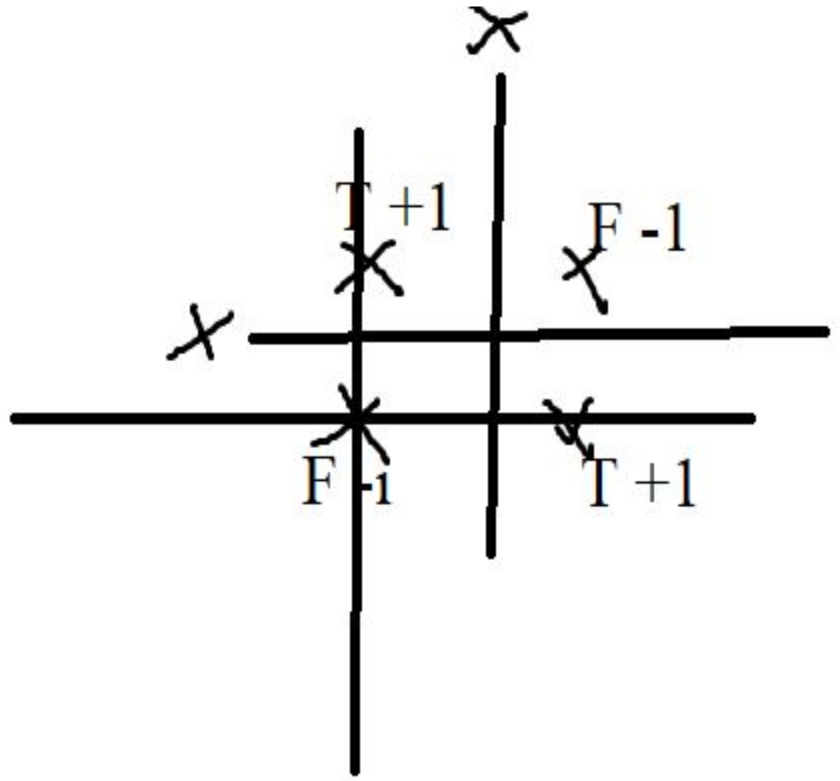


X	Y	
0	0	0
0	1	1
1	0	1
1	1	1

Linearly separable

Representational Power- Linearly non-separable

X-OR



x	y	
0	0	0
0	1	1
1	0	1
1	1	0

The Perceptron Training Rule

- Let us understand how to learn the weights for a single perceptron.
- The learning problem here is to determine a weight vector that causes the perceptron to produce the correct output (+ or - 1) for each of the given training example.
- Here we consider two algorithms:
    - the perceptron rule and
    - the delta rule
- These two algorithms provide the basis for learning networks of many units.

The Perceptron Training Rule

- We begin with random weights and iteratively apply the perceptron to each training example.
- The weights are modified whenever the perceptron misclassifies an example.
- According to the perceptron training rule the weight w_i associated with input x_i is updated according to the rule:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta(t - o)x_i$$

- Here t is the target output o is the output generated and η(=.1 generally) is a positive constant called the learning rate.
- Convergence is assured if the training examples are linearly separable.

Gradient Descent and the Delta Rule

- Visualizing the Hypothesis Space
- Derivation of the Gradient Descent Rule
- Stochastic Approximation to Gradient Descent

Gradient Descent and the Delta Rule

- The perceptron rule might fail to converge if the examples are not linearly separable.
- A rule called **delta rule** is designed to overcome this difficulty.
- The idea behind delta rule is to use **gradient descent** to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- The delta training rule is best understood by considering the task of training an un-thresholded perceptron, a linear unit for which the output o is given by:

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

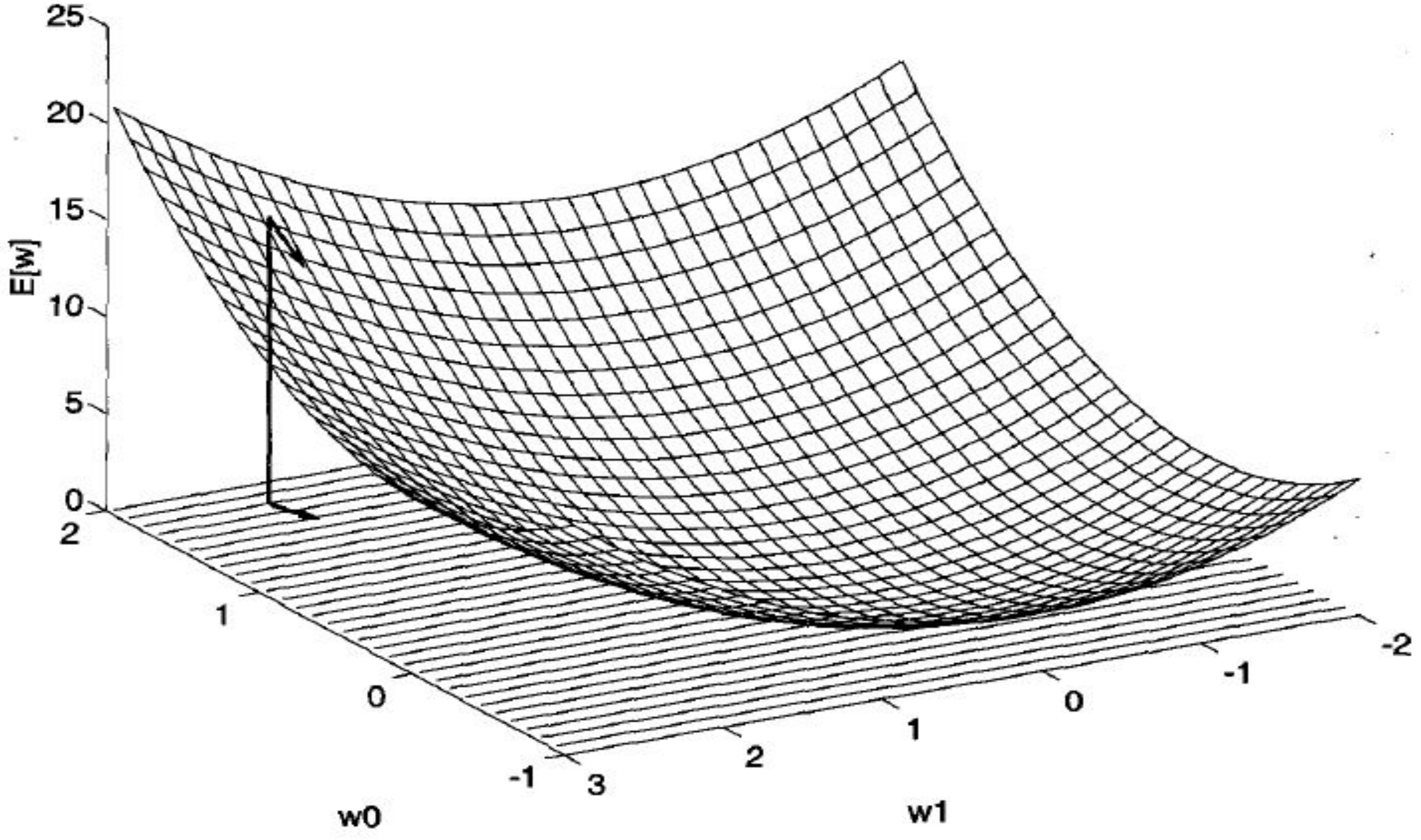
Gradient Descent and the Delta Rule

- We need to specify a measure for the training error of a hypothesis (weight vector) relative to the training examples in order to derive a weight learning rule for linear units.
- A common measure used mostly for the training error is:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

- Where D is the set of training examples t_d is the target output for training example d and o_d is the output of the linear unit for the training example d.

Visualizing the Hypothesis Space



Derivation of the Gradient Descent Rule

- The direction of the steepest descent can be found by computing the derivative of E with respect to each component of the vector w.
- The vector derivative is called the gradient of E with respect to the vector w. It is written as:

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots, \frac{\partial E}{\partial w_n} \right]$$

- $\nabla E(\vec{w})$ is a vector whose components are the partial derivatives of E with respect to each w_i.
- The gradient specifies the direction that produces the steepest increase in E. The negative of this vector gives the steepest decrease.

Derivation of the Gradient Descent Rule

- The training rule for gradient descent is:

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w}$$

where

$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$

- Here η is a positive constant called the learning rate which determines the step size in the gradient descent search. The negative sign moves the weight vector in the direction that decreases E.
- The training rule can also be written as:

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

Derivation of the Gradient Descent Rule

- The vector of $\frac{\partial E}{\partial w_i}$ derivatives that forms the gradient can be obtained by differentiating E from equation for training error:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2 \\
&= \frac{1}{2}\sum_{d\in D}2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d) \\
&= \sum_{d\in D}(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}\cdot\vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_{d\in D}(t_d - o_d)(-x_{id})
\end{aligned}$$

- Where x_{id} denotes the single input component x_i for example d.

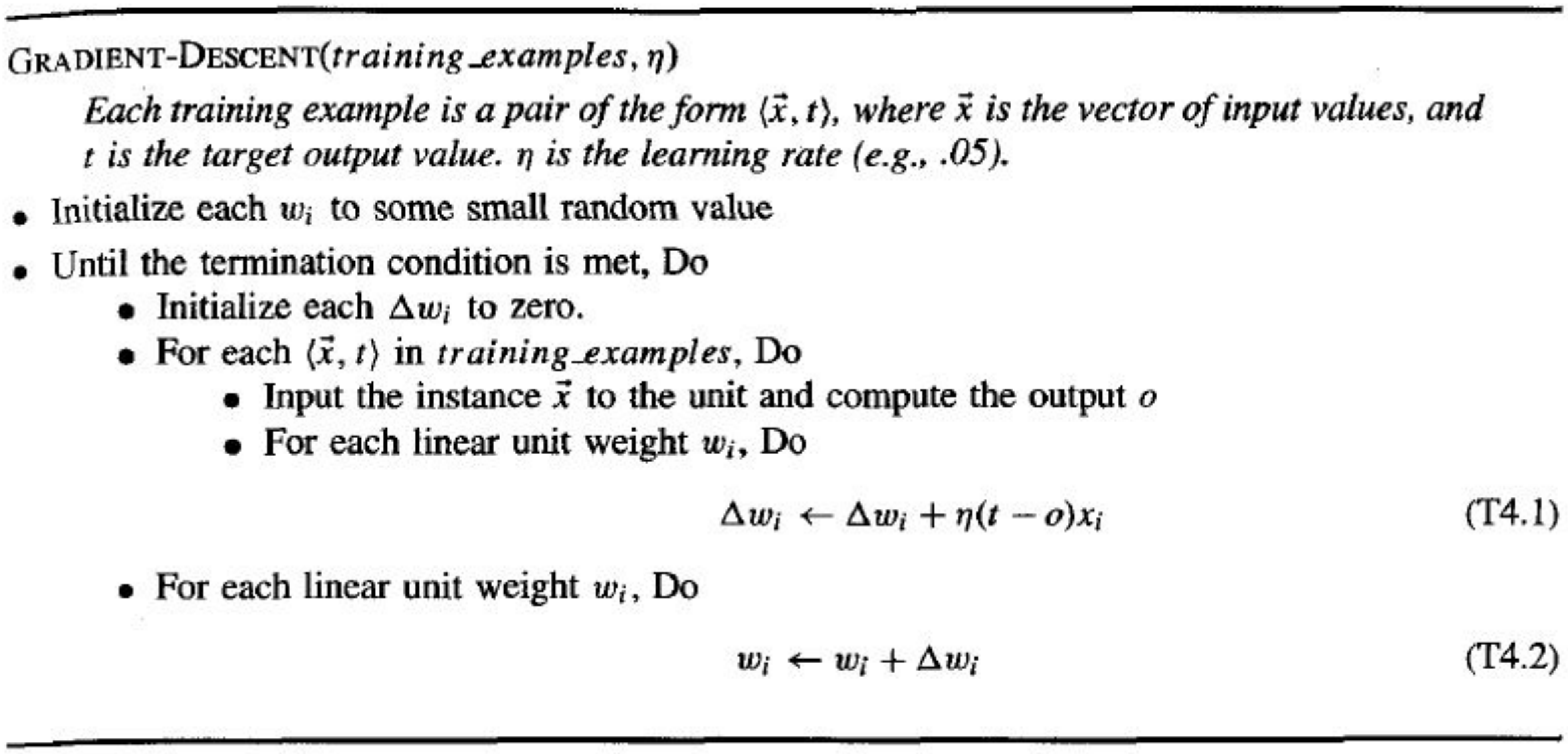
Derivation of the Gradient Descent Rule

- Now the weight update rule for gradient descent is:

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

- The gradient descent algorithm for training linear units is:
  - Pick an initial random weight vector
    - Apply the linear unit to all training examples
    - Compute Δw_i for each weight
  - Update each weight w_i by adding Δw_i

Derivation of the Gradient Descent Rule



```
GRADIENT-DESCENT(training_examples, $\eta$)
    Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and
    $t$ is the target output value. $\eta$ is the learning rate (e.g., .05).
• Initialize each $w_i$ to some small random value
• Until the termination condition is met, Do
    • Initialize each $\Delta w_i$ to zero.
    • For each $\langle \vec{x}, t \rangle$ in training_examples, Do
        • Input the instance $\vec{x}$ to the unit and compute the output $o$
        • For each linear unit weight $w_i$, Do
              $\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$        (T4.1)
    • For each linear unit weight $w_i$, Do
              $w_i \leftarrow w_i + \Delta w_i$        (T4.2)
```

Stochastic Approximation to Gradient Descent

- Gradient Descent is a strategy for searching through a large or infinite hypothesis space that can be applied whenever:
  - The hypothesis space contains continuously parameterized hypotheses.
  - The error can be differentiated with respect to these hypothesis parameters.
- The key practical difficulties in applying gradient descent are:
  - Converging to a local minimum can sometimes be quite slow
  - If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minima.

Stochastic Approximation to Gradient Descent

- One common variation on gradient descent intended to overcome these difficulties is called **incremental gradient descent,** or alternatively **stochastic gradient descent.**
- In gradient descent training rule the weights are updated after summing over all the training examples in D.
- The algorithm for stochastic gradient descent removes the equation T 4.2 in the algorithm for gradient descent and updates the equation T 4.1 as (incremental update of weights):

$$w_i \leftarrow w_i + \eta(t - o)\, x_i$$

- The error function can be defined for each individual example as follows:

$$E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$$

Stochastic Approximation to Gradient Descent

- The key differences between standard gradient descent and stochastic gradient descent are:
  - In standard gradient descent, the error is summed over all examples before updating weights, whereas in stochastic gradient descent weights are updated upon examining each training example.
  - Standard gradient descent is often used with a larger step size per weight update than stochastic gradient descent.
  - Stochastic gradient descent can sometimes avoid falling into these local minima because it uses the $\nabla E_d(\vec{w})$ rather than $\nabla E(\vec{w})$ to guide its search.

Stochastic Approximation to Gradient Descent

- The rules for the perceptron training rule and the delta rule seem to same but the difference is in the delta rule the o refers to the linear unit output $o(\vec{x}) = \vec{w} \cdot \vec{x}$ where as for the perceptron rule o refers to the thresholded output $o(\vec{x}) = sgn(\vec{w} \cdot \vec{x})$.
- Delta rule can also be used to train for thresholded perceptron units.

Remarks

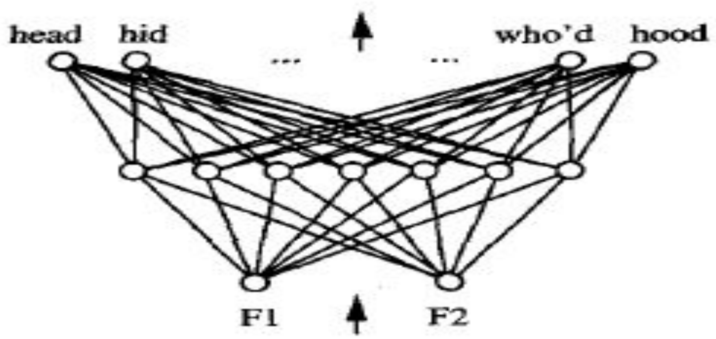
- The perceptron training rule updates the weights based on the error in the thresholded perceptron output, where as the delta rule updates weights based on the error in the unthresholded linear combination of inputs.
- The perceptron training rule converges after a finite number of iterations to a hypothesis that perfectly classifies the training data provided the training examples are linearly seperable.
- The delta rule converges only asymptotically towards the minimum error hypothesis possibly requiring unbounded times, but converges regardless of whether the training data are linearly separable.
- Another possible algorithm is linear programming which solves sets of linear inequalities (cannot be extended to multilayer networks).

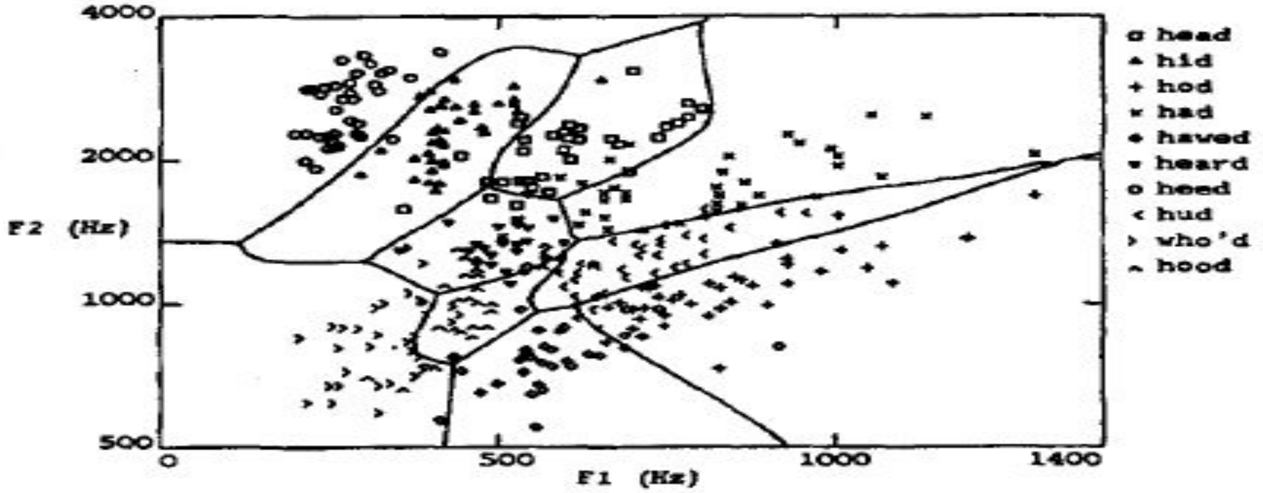
Multilayer Networks and the Backpropogation Algorithm

- A differentiable threshold unit
- The backpropogation algorithm
- Derivation of the backpropogation rule

Multilayer Networks and the Backpropogation Algorithm

- Single perceptrons can express only linear decision surfaces, where as multilayer networks learned by backpropogation algorithm are capable of expressing a rich variety of nonlinear decision surfaces.



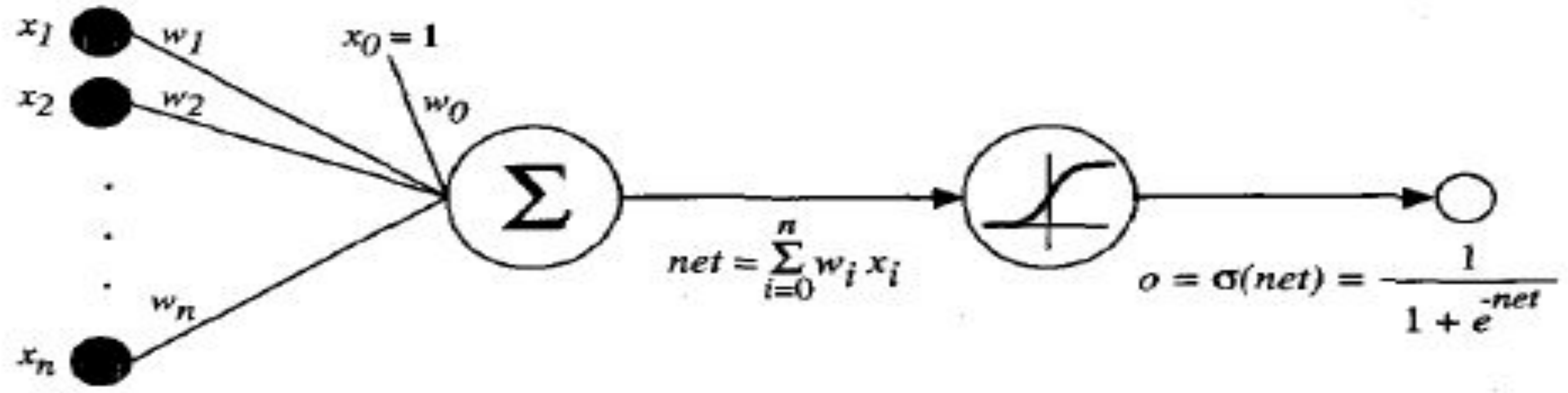


- The two parameters F1 and F2 are obtained from a spectral analysis of the sound.

A Differentiable Threshold Unit

- Can we use the linear units discussed earlier for which gradient descent is derived as the learning rule as the basis for construction of multilayer networks? (still produce only linear units).
- Can we use the perceptron unit as the base for multilayer networks? (the discontinuous threshold makes it un-differentiable).
- We need a unit whose output is:
    - A non-linear function of its input
    - Output is a differentiable function of its input
- A **sigmoid unit** which is a smoothed differentiable threshold function is a solution.

A Differentiable Threshold Unit



x_1
x_2
x_n
w_1
w_2
w_n
$x_0 = 1$
w_0
Σ
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

The sigmoid unit computes its output o as:

$$o = \sigma(\vec{w} \cdot \vec{x})$$

where

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

σ is called the sigmoid function or the logistic function

A Differentiable Threshold Unit

- The output of the sigmoid function ranges between 0 and 1 increasing monotonically with its input.
- It maps a very large input domain to a small range of outputs and therefore it is referred to as the squashing function of the unit.
- It has a very useful property that its derivative can be expressed in terms of its output.
- The gradient descent learning rule makes use of the derivative.
- Other functions that can be used in place of sigmoid function are:
    - e^{-y} can sometimes be replaced by $e^{-k.y}$ where k determines the steepness of the threshold.
    - tanh can also be used sometimes.

The Backpropogation Algorithm

- Adding Momentum
- Learning in Arbitrary Acyclic Networks
- Derivation of the Backpropogation Rule

The Backpropogation Algorithm

- The backpropogation algorithm learns the weights for a multilayer network, given a network with a fixed set of units and interconnections.
- Gradient descent is used to minimize the squared error between the network output values and the target values of these outputs.
- Here E is redefined to sum the errors over all of the network output units:

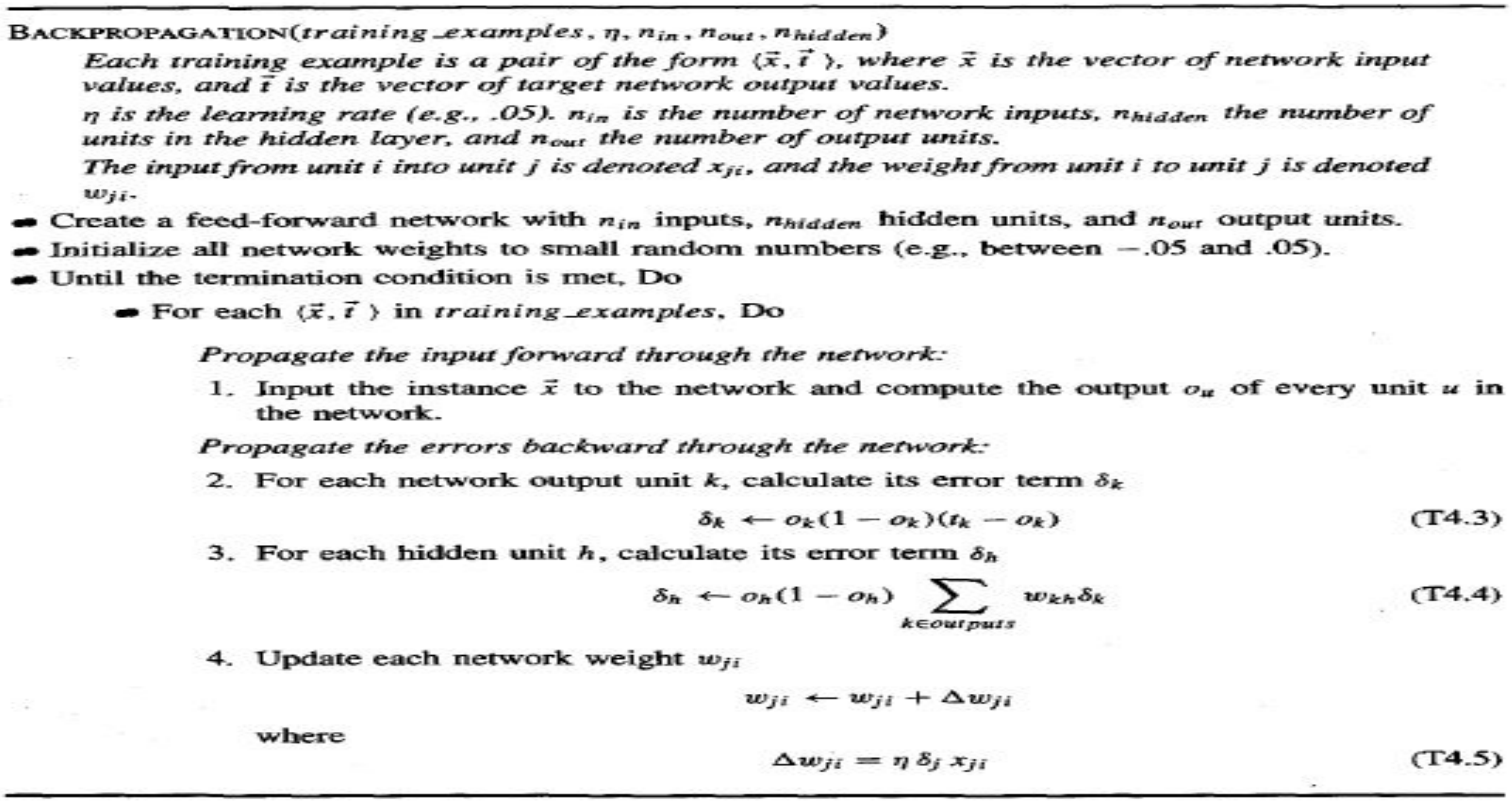
$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2$$

- Where outputs is the set of output units in the network, and t_{kd} and o_{kd} are the target and output values associated with the kth output unit and training example d.

The Backpropogation Algorithm

- The learning problem faced by backpropogation is to search a large hypothesis space defined by all possible weight values for all the units in the network.
- Here also gradient descent can be used to attempt to find a hypothesis to minimize E.
- In multilayer network the error surface can have multiple local minima, but still backpropogation has been successful in many real world applications.

The Backpropogation Algorithm



```
BACKPROPAGATION(training_examples, $\eta$, $n_{in}$, $n_{out}$, $n_{hidden}$)
    Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input
    values, and $\vec{t}$ is the vector of target network output values.
    $\eta$ is the learning rate (e.g., .05). $n_{in}$ is the number of network inputs, $n_{hidden}$ the number of
    units in the hidden layer, and $n_{out}$ the number of output units.
    The input from unit i into unit j is denoted $x_{ji}$, and the weight from unit i to unit j is denoted
    $w_{ji}$.
• Create a feed-forward network with $n_{in}$ inputs, $n_{hidden}$ hidden units, and $n_{out}$ output units.
• Initialize all network weights to small random numbers (e.g., between $-.05$ and $.05$).
• Until the termination condition is met, Do
    • For each $\langle \vec{x}, \vec{t} \rangle$ in training_examples, Do
        Propagate the input forward through the network:
        1. Input the instance $\vec{x}$ to the network and compute the output $o_u$ of every unit $u$ in
           the network.
        Propagate the errors backward through the network:
        2. For each network output unit $k$, calculate its error term $\delta_k$
               $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$                        (T4.3)
        3. For each hidden unit $h$, calculate its error term $\delta_h$
               $\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k$        (T4.4)
        4. Update each network weight $w_{ji}$
               $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
           where
               $\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$                             (T4.5)
```

The Backpropogation Algorithm

- The algorithm applies to layered feed forward networks containing two layers of sigmoid units with units at each layer connected to all units from the preceding layer.
- The notion used here is the same as that used in earlier sections with the following extensions:
  - An index is assigned to each node in the network, where a "node" is either an input to the network or the output of some unit in the network.
  - x_{ji} denotes the input from node i to unit *j* **,** and w_{ji} denotes the corresponding weight.
  - δ_{n} denotes the error term associated with unit ***n.*** It plays a role analogous to the quantity *(t - o)* in our earlier discussion of the delta training rule.

The Backpropogation Algorithm

- The difference between the weight update rule in the current algorithm and the delta rule is that the error (t-o) in delta rule is replaced by δ_j.
- δ_k is obtained by multiplying (t_k-o_k) by the factor $o_k(1-o_k)$ which is the derivative of the sigmoid function.
- Since no target values are provided for hidden layers the error term δ_h for the hidden unit is calculated by summing the error term δ_k for each output unit influenced by h weighting each of the δ_k's by w_{kh}.
- The weight characterizes the degree to which hidden unit h is responsible for the error in output unit k.

Adding Momentum

- Among the variations that have been developed for backpropogation the most common is to alter the weight update rule in the algorithm by making the weight update on the nth iteration depend partially on the update that occurred during the (n-1)th iteration.

$$\Delta w_{ji}(n) = \eta\, \delta_j\, x_{ji} + \alpha \Delta w_{ji}(n-1)$$

- $\Delta\, w_{ji}(n)$ is the weight update performed during the nth iteration through the main loop of the algorithm, $0 <= \alpha < 1$ is a constant called the momentum.
- The second term in the right side of the equation is called as momentum term.

Learning in Arbitrary Acyclic Networks

- The backpropagation algorithm presented here is for two layer network which can be easily generalized to feedforward networks of arbitrary depth.
- The only change is to the procedure for computing δ values.
- The δ_r value for a unit r in layer m is computed from the δ values at the next deeper layer m+1 according to :

$$\delta_r = o_r\,(1 - o_r) \sum_{s \in layer\, m+1} w_{sr}\,\delta_s$$

- In case the network units are not arranged in uniform layers:

$$\delta_r = o_r\,(1 - o_r) \sum_{s \in Downstream(r)} w_{sr}\,\delta_s$$

- Downstream(r) is the set of units immediately downstream from r.

Derivation of the Backpropogation Rule

- Here we discuss about the derivation of the backpropogation weight-tuning rule.
- The stochastic gradient descent involves iterating through the training examples one at a time.
- For each training example d every weight w_{ii} is updated by adding to it $\Delta w_{ji.}$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

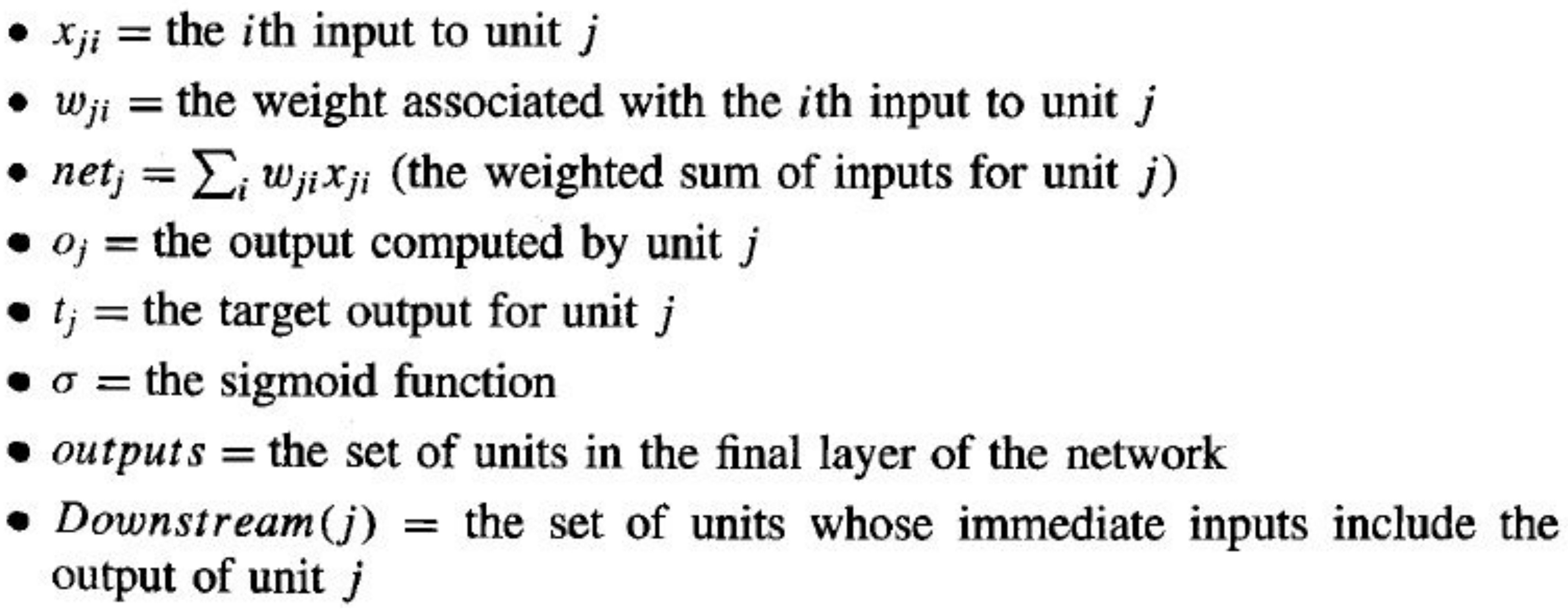
- where E_d is the error on the training example d, summed over all output units in the network

$$E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

- Outputs is the set of outputs in the network

Derivation of the Backpropogation Rule

- The notations used in the derivation are as follows:



- x_{ji} = the ith input to unit j
- w_{ji} = the weight associated with the ith input to unit j
- $net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)
- o_j = the output computed by unit j
- t_j = the target output for unit j
- σ = the sigmoid function
- $outputs$ = the set of units in the final layer of the network
- $Downstream(j)$ = the set of units whose immediate inputs include the output of unit j

Derivation of the Backpropogation Rule

- We now derive an expression for $\frac{\partial E_d}{\partial w_{ji}}$ in order to implement the stochastic gradient descent rule.
- The weight w_{ji} can influence the rest of the network only through net_j. We can therefore use the chain rule to write:

$$\begin{aligned}\frac{\partial E_d}{\partial w_{ji}} &= \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} \\ &= \frac{\partial E_d}{\partial net_j} x_{ji}\end{aligned}$$

- The remaining task is to derive a convenient expression for $\frac{\partial E_d}{\partial net_j}$. There are two cases here:
  - Training rule for output unit weights
  - Training rule for hidden unit weights

Training Rule for Output Unit Weights

- net$_j$ can influence the network only through o$_j$. We can invoke the chain rule to write:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

- The first term in the equation

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

- The derivatives will be 0 for all the output units k except when k=j.

$$\begin{aligned}\frac{\partial E_d}{\partial o_j} &= \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2 \\ &= \frac{1}{2}2(t_j - o_j)\frac{\partial (t_j - o_j)}{\partial o_j} \\ &= -(t_j - o_j)\end{aligned}$$

Training Rule for Output Unit Weights

- Now let us consider the second term in the equation. Since o$_j$=$\sigma(net_j)$ The derivative is just the derivative of the sigmoid function which is equal to $\sigma(net_j)$(1-$\sigma(net_j)$)

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$
$$= o_j(1 - o_j)$$

- Now we get

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\ o_j(1 - o_j)$$

- The stochastic gradient descent rule for the output units is:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta\ (t_j - o_j)\ o_j(1 - o_j)x_{ji}$$

Training Rule for Hidden Unit Weights

- In the case where j is an internal or hidden unit in the network the derivation of the training rule for w_{ji} must consider the indirect ways in which w_{ji} can influence the network outputs and hence E_d.
- We refer to the set of all units immediately downstream of unit j in the network.
- This set of units is denoted by Downstream(j).
- net_j can influence the network outputs only through the units in Downstream(j).

Training Rule for Hidden Unit Weights

- We can write:

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \; w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{k \in Downstream(j)} -\delta_k \; w_{kj} \; o_j(1 - o_j)
\end{aligned}$$

- Using δj for - $\frac{\partial E_d}{\partial net_j}$

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \; w_{kj}$$

and

$$\Delta w_{ji} = \eta \; \delta_j \; x_{ji}$$

Remarks on the Backpropogation Algorithm

- Convergence and Local Minima
- Representational Power of Feedforward networks
- Hypothesis space search and inductive bias
- Hidden Layer Representation
- Generalization, Overfitting and Stopping Criteria

Convergence and Local Minima

- The error surface of multilayer networks may contain many different local minima and therefore gradient descent may get trapped in any of them.
- In many practical applications the problem of local minima has not been found to be as severe as we might think.
- In cases where the error surfaces are high dimensional, if the gradient descent falls into local minima with respect to one weight it will not be the local minima with respect to other weights.

Convergence and Local Minima

- The sigmoid threshold function is approximately linear when the weights are close to zero and as the weights grow it starts behaving non-linearly.
- We can expect the local minima to exist in the region of the weight space that represents more complex functions (non-linear).
- Once these points are reached they will be very close to the global minima and they can be acceptable.

Convergence and Local Minima

- Common heuristics to attempt to alleviate the problem of local minima include:
  - Add a momentum term to the weight update rule as discussed already.
  - Use stochastic gradient descent rather that gradient descent.
  - Train multiple networks using the same data but initialize each network with different random weights.

Representational Power of Feedforward Networks

- The answer to which functions can be represented by feedforward networks depends on the width and depth of the networks. A few functions that can be represented are:
    - **Boolean functions.** Every Boolean function can be represented exactly by some network with two layers of units, although the number of hidden units required grows exponentially in the worst case with the number of network inputs.
    - **Continuous functions.** Every bounded continuous function can be approximated with arbitrarily small error (under a finite norm) by a network with two layers of units.
    - **Arbitrary functions.** Any function can be approximated to arbitrary accuracy by a network with three layers of units.

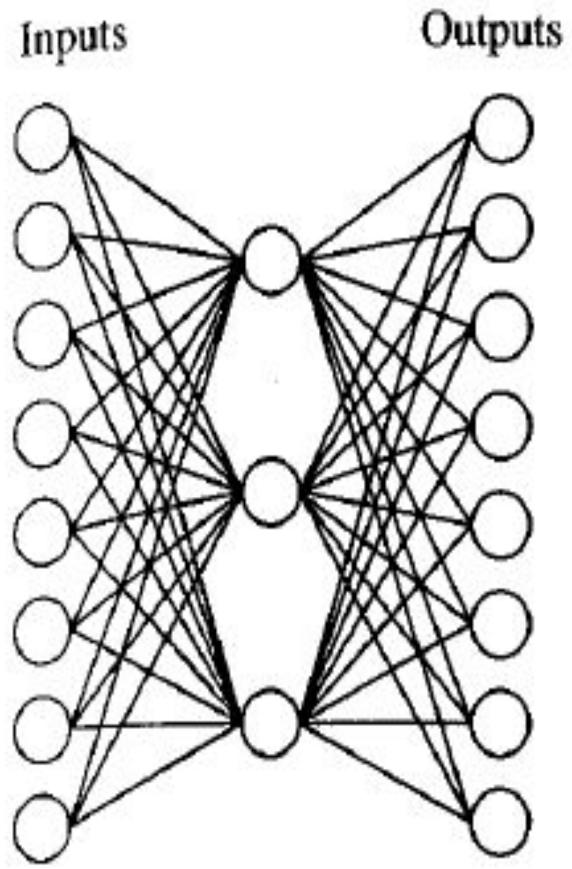
Hypothesis Space Search and Inductive Bias

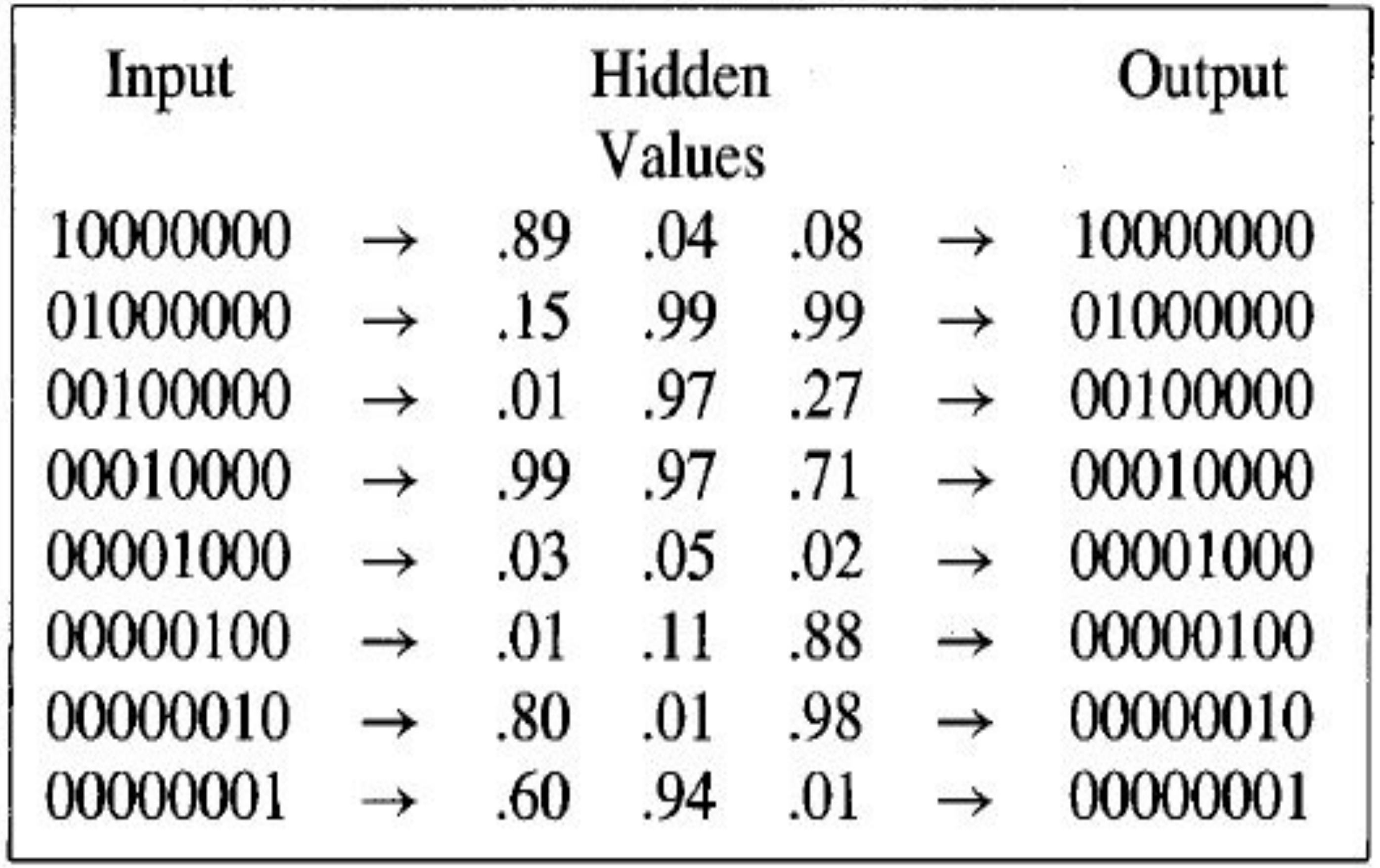
- The hypothesis space in backpropogation is continuous in contrast to the hypothesis space of decision tree learning based on discrete representations.
- The well defined error gradient because of the differentiable continuous hypothesis space provides a very useful structure for organizing the search for the best hypothesis.
- Backpropogation will tend to label points in between two positive training examples with no negative examples between them as positive examples as well.

Hidden Layer Representations

- A very important property of backpropogation is its ability to discover useful intermediate representations at the hidden unit layer inside the network.
- The weight tuning procedure in backpropogation is free to set weights that define whatever hidden unit representation is most effective at minimizing the squared error E.
- It defines new hidden layer features that can capture properties of the input instances that are most relevant to learning the target function.

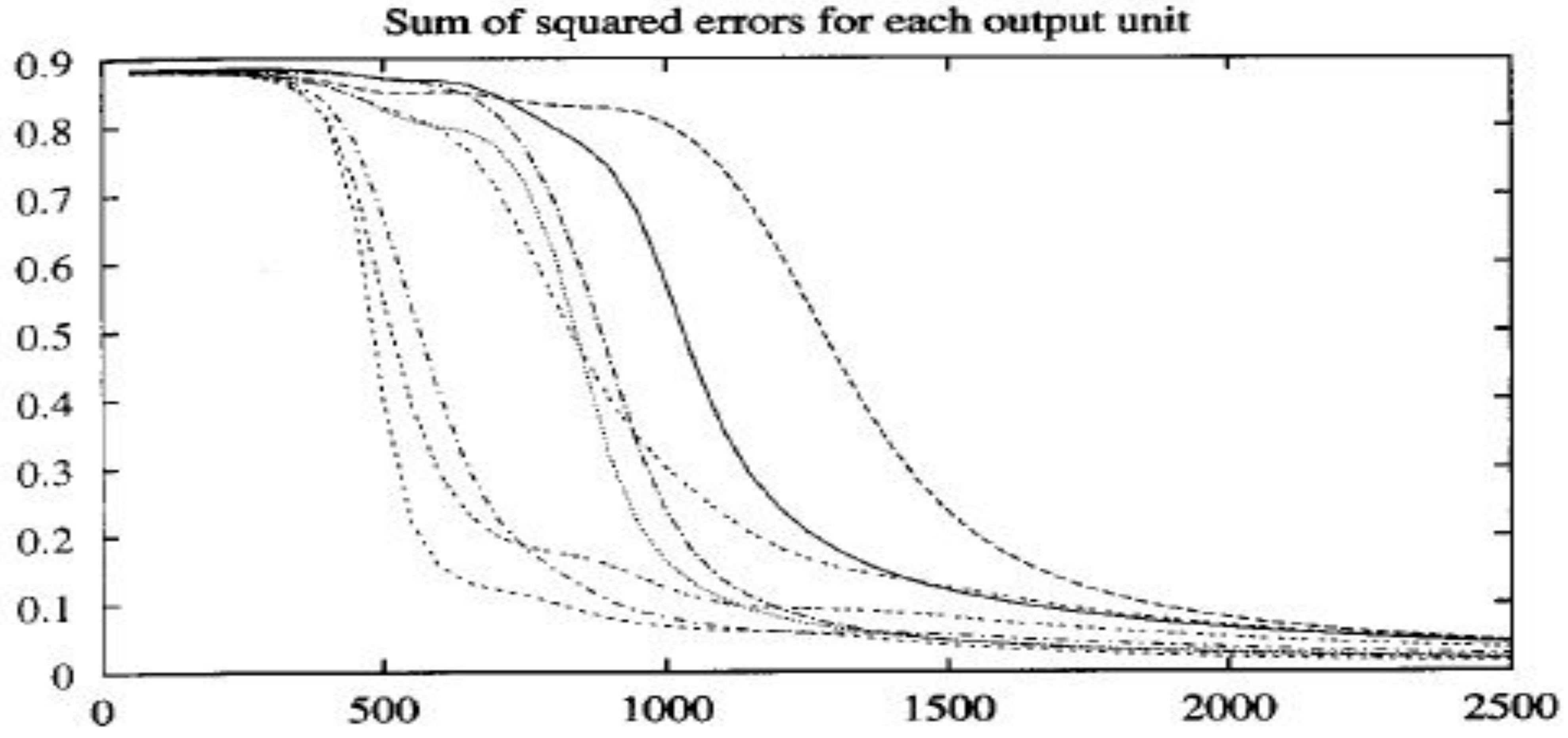
Hidden Layer Representations



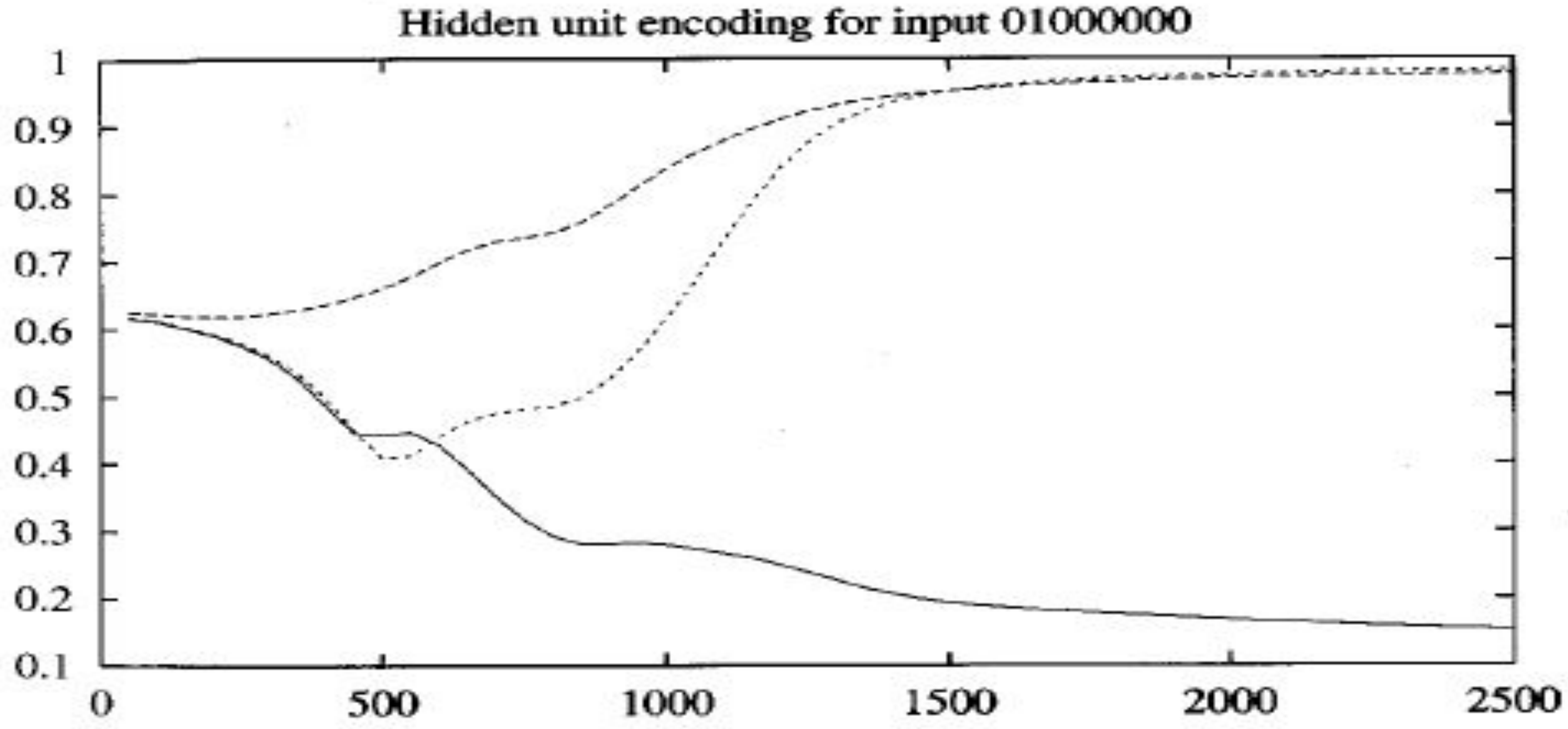


Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

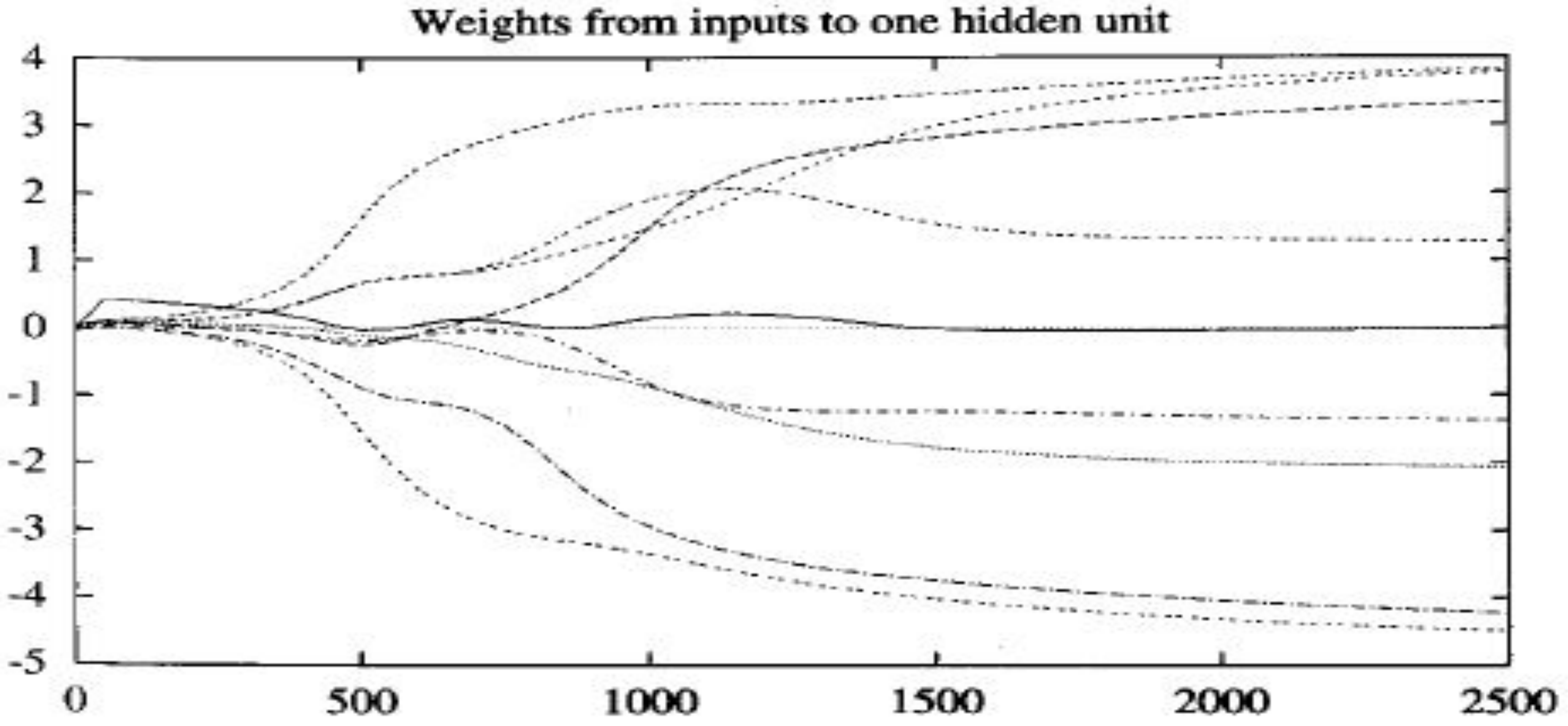
Hidden Layer Representations



Hidden Layer Representations



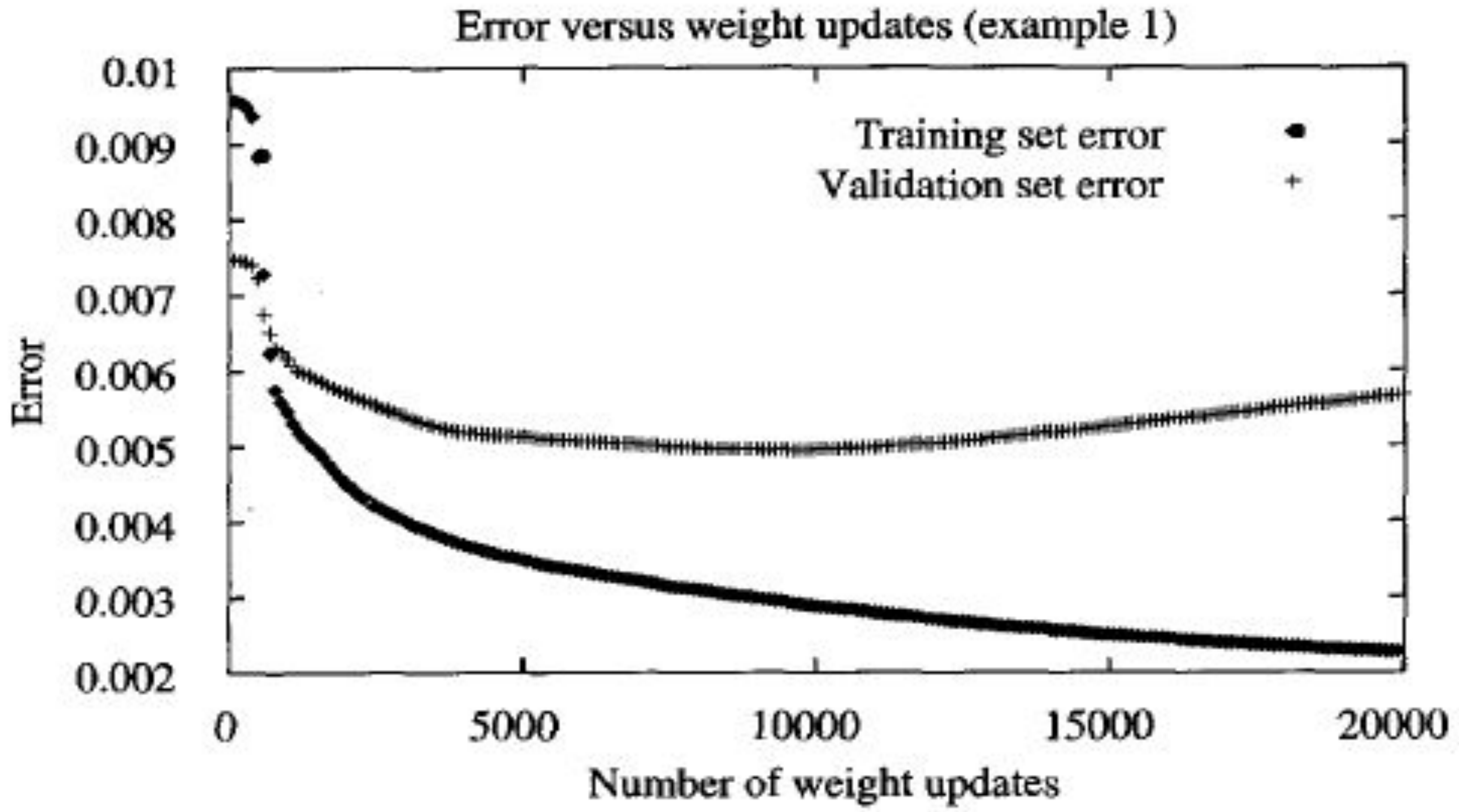
Hidden Layer Representations



Generalization, Overfitting and Stopping Criteria

- The most important question in backpropogation is, what is an appropriate termination condition?
- One obvious choice will be to wait until the error on training examples falls below a predefined threshold and then stop.
- This might result in overfitting the training examples.

Generalization, Overfitting and Stopping Criteria

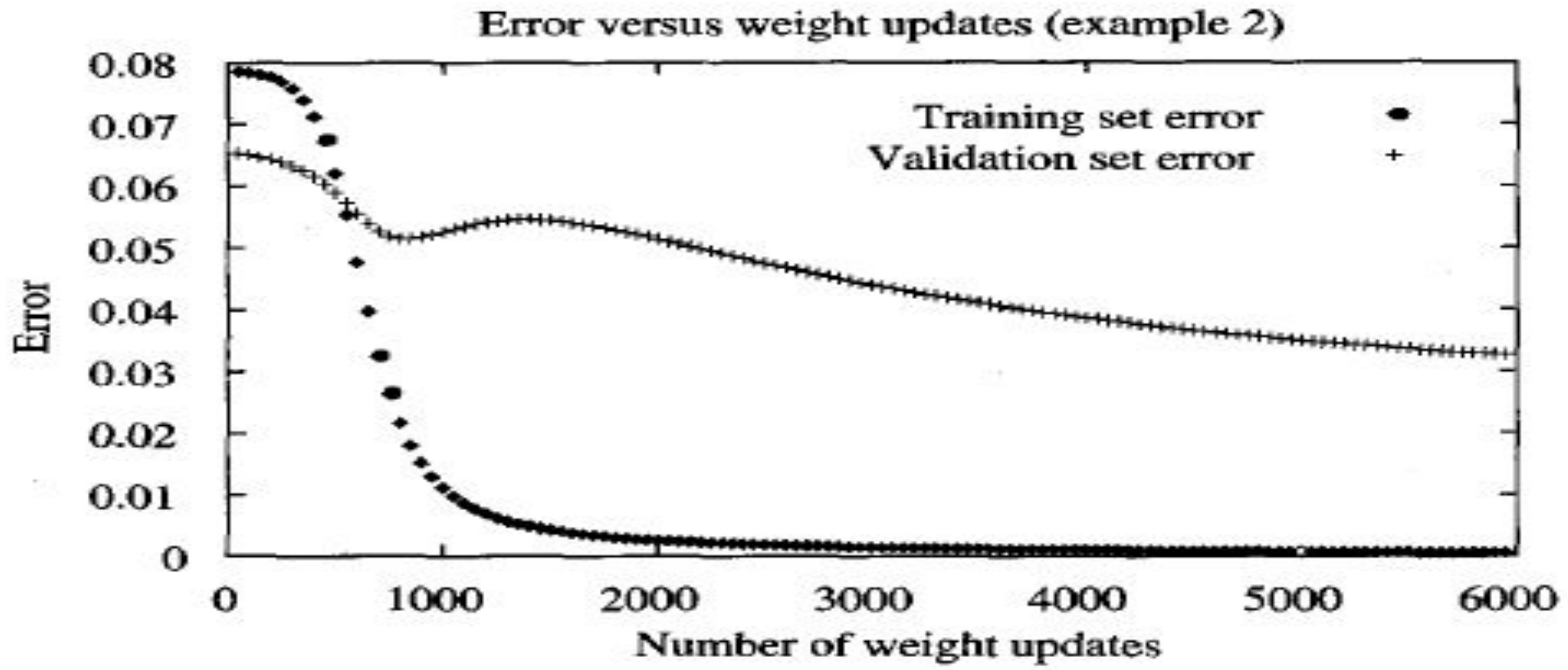


The above line in the figure measures the generalization accuracy of the network- the accuracy with which it fits the examples beyond the training data.

Generalization, Overfitting and Stopping Criteria

- Why does overfitting tend to occur during the later iterations but not during the earlier iterations?
- Several techniques are available to overcome the overfitting problem of backpropogation:
  - Weigh decay-- it decreases each weight by some small factor during each iteration. The motivation for this approach is to keep weight values small, to bias learning against complex decision surfaces.
  - Providing a validation set in addition to the training data. The validation set also might have local minima and global minima and sufficient care must be taken about it.
- When to data set is small we need to divide the set into training and validation set and train by exchanging the sets.

Generalization, Overfitting and Stopping Criteria



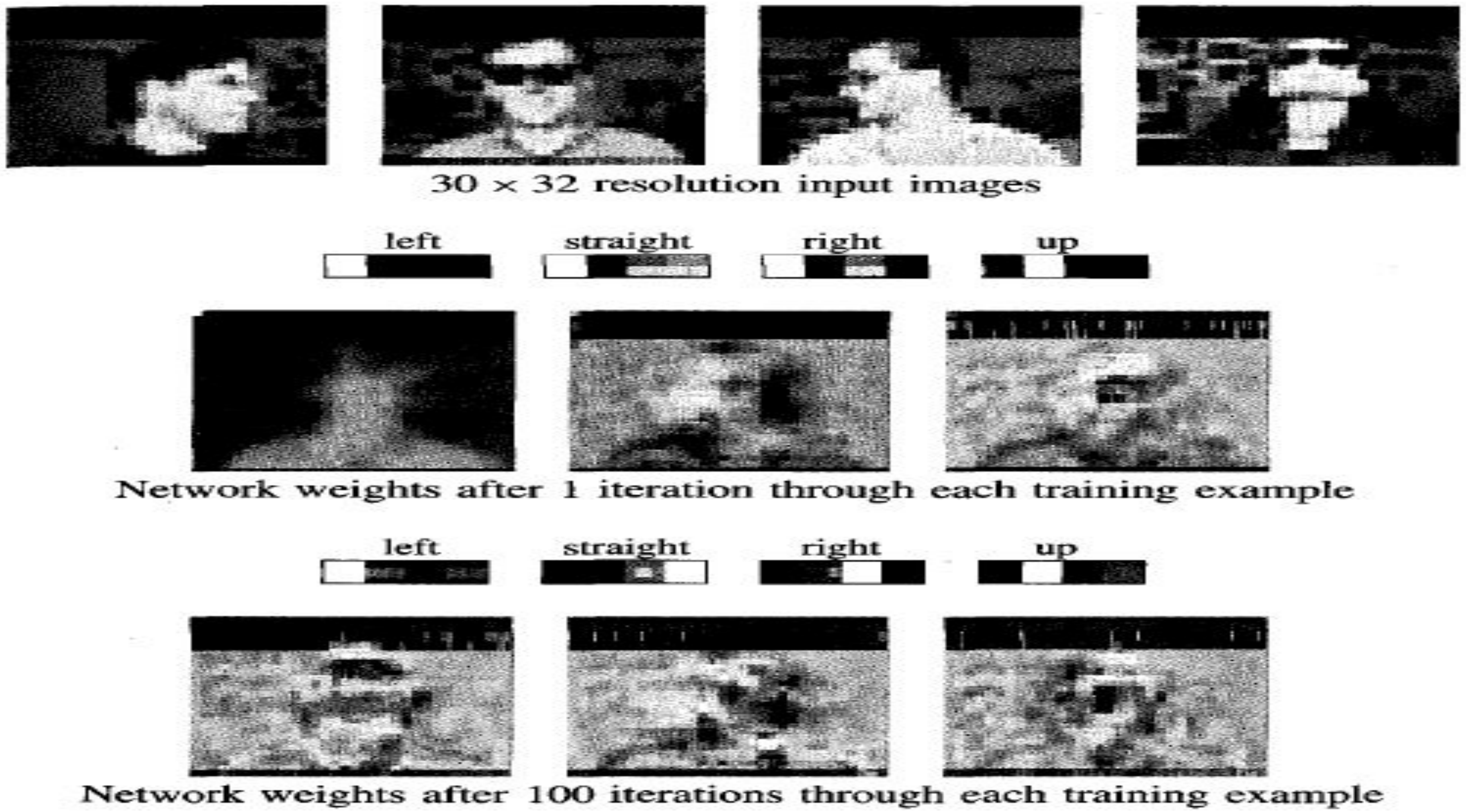
An Illustrative Example: Face Recognition

- The Task
- Design Choices
- Learned Hidden Representations

The Task

- The learning task involves classifying images of 20 different people including approximately 32 images per person by varying the person’s expressions the direction in which they were looking and whether or no they were wearing glasses.
- A total of 624 greyscale images were collected, each with a resolution of 120*128, with each image pixel described by a greyscale intensity value between 0 and 255.
- Here we take up the task of the direction in which the person is looking (left, right, ahead, up).

The Task


30 × 32 resolution input images
left
straight
right
up
Network weights after 1 iteration through each training example
left
straight
right
up
Network weights after 100 iterations through each training example

Design Choices

- Input Encoding
- Output Encoding
- Network Graph Structure
- Other learning algorithm parameters

Input Encoding

- The input to the ANN must be some representation of the image. How do we encode the image?
- The image can be preprocessed to extract the edges, regions of uniform intensity, or other local image features, then input these features to the network.
- The design option chosen was to encode the image as a 30*32 (a coarse resolution summary of the original 120*128 captured image) pixel intensity value with one network input per pixel.
- The pixel intensity value which ranges from 0 to 255 were linearly scaled to a range of 0 to 1.

Output Encoding

- The ANN must output one of the four values indicating the direction in which the person is looking (left, right, up or straight).
- The encoding can be done using a single output unit assigning the values of 0.2, 0.4, 0.6, 0.8 to encode these four possible values.
- The output layer was designed in such a way that four distinct output were used each representing one direction. The one with the highest valued output is considered as the direction of the face.
- This is called as 1 of n output encoding. The reasons for using this type of encoding are:
  - It provides more degree of freedom to the network for representing the target function.
  - The difference in the highest value and the second highest can be used as a measure of confidence in the network prediction.

Output Encoding

- Another design choice here is what should be the values for each of the four output units?
- One choice would be to place 1 in one of the output and 0 in the others. The one with the 1 output is considered as the direction.
- Instead of 0 and 1 the values 0.1 and 0.9 are used.
- These values are used because the sigmoid unit can produce these values given finite weights.

Network Graph Structure

- As we already know backpropogation can be applied to any acyclic directed graph of sigmoid units.
- The next design choice is how many units to use and how to interconnect them.
- Here the choice is a standard feed forward network with every unit in one layer connected to every unit in the next layer.
- The are two layers in addition to the input layer, one hidden layer and one output layer.
- Next question is how many hidden units should be included?
- It has been found in many applications that some minimum number of units are required to learn the target function. There units are used in this case.

Other Learning Algorithm Parameters

- The learning rate was set to .3 and the momentum was set to .3. These lower values produced equivalent generalization accuracy but longer training times.
- Gradient descent was used in all the experiments.
- The weights in the output units were initialized to random values while the weights in the input were initialized to 0.
- The data was partitioned into training set and validation set.
- After every 50 gradient descent steps the validation was performed on the validation set.
- The final report accuracy was measured over a third set of examples.

Learned Hidden Representations

- It will be very interesting to examine the learned weight values for the 2899 weights in the network.
- The four rectangular boxes in the figure depict the weights of the output units.
- The four squares within each rectangle indicate the four weights associated with the output unit.

Advanced Topics in ANN

- Alternative Error Functions
- Alternative Error Minimization Procedures
- Recurrent Networks
- Dynamically modifying network structure

Alternative Error Functions

- Other definitions have been suggested for calculation of E in order to incorporate other constraints into the weight-tuning rule. Examples of alternative definitions of E include:
- Adding a penalty term with the weight magnitude

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

which yields a weight update rule where each weight is multiplied by the constant (1-2γη) upon each iteration. Choosing this definition of E is equivalent to using a weight decay strategy.

Alternative Error Functions

- Adding a term for error in the slope or derivative of the target function.

$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d\in D}\sum_{k\in outputs}\left[(t_{kd}-o_{kd})^2+\mu\sum_{j\in inputs}\left(\frac{\partial t_{kd}}{\partial x_d^j}-\frac{\partial o_{kd}}{\partial x_d^j}\right)^2\right]$$

where x^j_d denotes the jth input value for the training example d.

$\frac{\partial t_{kd}}{\partial x_d^j}$ is the training derivative describing how the target output value t_{kd} should vary with respect to x^j_d

$\frac{\partial o_{kd}}{\partial x_d^j}$ is the corresponding derivative of the actual learned network.

μ is the relative weight placed on fitting the training values versus training derivatives.

Alternative Error Functions

- Minimizing the cross entropy of the network with respect to the target values.

$$-\sum_{d \in D} t_d \log o_d + (1 - t_d) \log(1 - o_d)$$

Here o_d is the probability estimate output by the network for training example d, and t_d is the 1 or 0 target value for training example d.

- Altering the effective error function can also be accomplished by weight sharing, or "tying together" weights associated with different units or inputs.
- The idea here is that different network weights are forced to take on identical values, usually to enforce some constraint known in advance to the human designer.

Alternative Error Minimization Procedure

- Gradient descent is one of the most general search methods for finding a hypothesis to minimize the error function, but not the most effective.
- The weight update method here involves two decisions:
  - Choosing the direction in which to alter the current weight vector.
  - Choosing a distance to move.
- One optimization method known as line search involves a different approach to choosing the distance for the weight update.
- Once a line is chosen that specifies the direction of the update, the update distance is chosen by finding the minimum of the error function along this line.

Alternative Error Minimization Procedure

- Another method that builds on the idea of line search is called the conjugate gradient method.
- Here a sequence of line searches is performed to search for a minimum in the error surface.
- The first step in the sequence choses a negative gradient and in the subsequent steps a new direction is chosen.

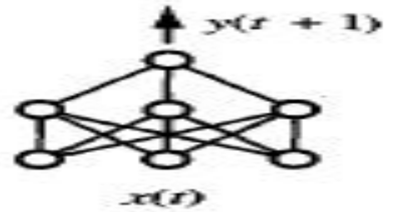
Recurrent Networks

- Until now we considered only network topologies that correspond to acyclic directed graphs.
- Recurrent networks are artificial neural networks that apply to time series data and that use outputs of network units at time t as the input to other units at time t+1.
- Given a time series of data the limitation of a feed forward network is that the prediction of y(t+1) depends only on x(t) and will not be able to capture the earlier values of x.
- To over come this difficulty a new unit b is added to the hidden and a new input unit c(t).The value of c(t) is defined as the value of unit b at time t-1.

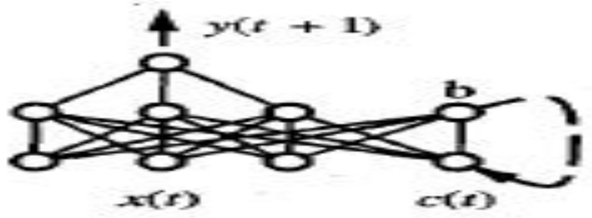
Recurrent Networks

- To train the recurrent networks let us consider the unfolded network in the diagram of the next slide.
- The weights in the unfolded network can be trained directly by using backpropagation.
- After training the unfolded network the fine weight w_{ji} can be taken as the mean of the corresponding weights in the various copies.

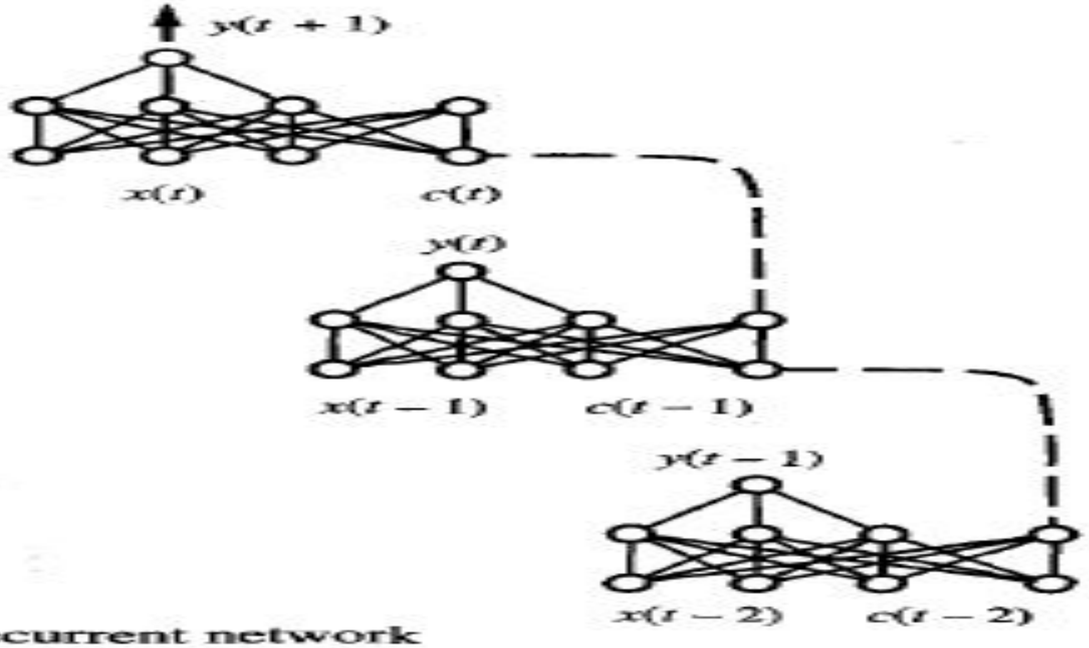
Recurrent Networks



(a) Feedforward network



(b) Recurrent network



(c) Recurrent network unfolded in time

Dynamically Modified Network Structure

- The cascade-correlation algorithm begins with a network with no hidden layer and grows the network as and when required.
- In case the residual error is more and the target function cannot be learnt a hidden layer is added.
- Another approach is to start with a very complex network and slowly prune the hidden units to get the appropriate result.
- This approach is called as “optimal brain damage” approach.

Evaluation Hypothesis

- Motivation
- Estimating Hypothesis Accuracy
- Basics of Sampling Theory
- A General Approach for Deriving Confidence Intervals
- Difference in Error of Two Hypothesis
- Comparing Learning Algorithms

Evaluation Hypothesis

- The discussion here is an introduction to statistical methods for estimating hypothesis accuracy focusing on three questions:
  - Given the observed accuracy of a hypothesis over a limited sample data, how well does this estimate its accuracy over additional examples?
  - Given that one hypothesis outperforms another over some sample of data, how probable is it that this hypothesis is more accurate in general?
  - When data is limited what is the best way to use this data to both learn a hypothesis and estimate its accuracy?

Motivation

- It is important to evaluate the performance of learned hypothesis as precisely as possible.
- When we must learn a hypothesis and estimate its future accuracy given only a limited set of data, two key difficulties arise:
  - Bias in the estimate
  - Variance in the estimate
- Here we discuss methods for:
  - Evaluating learned hypothesis.
  - Methods for comparing the accuracy of two hypothesis.
  - Methods for comparing the accuracy of two learning algorithms when only limited data is available.

Estimating Hypothesis Accuracy

- Sample Error and True Error
- Confidence Intervals for discrete valued hypothesis

Estimating Hypothesis Accuracy

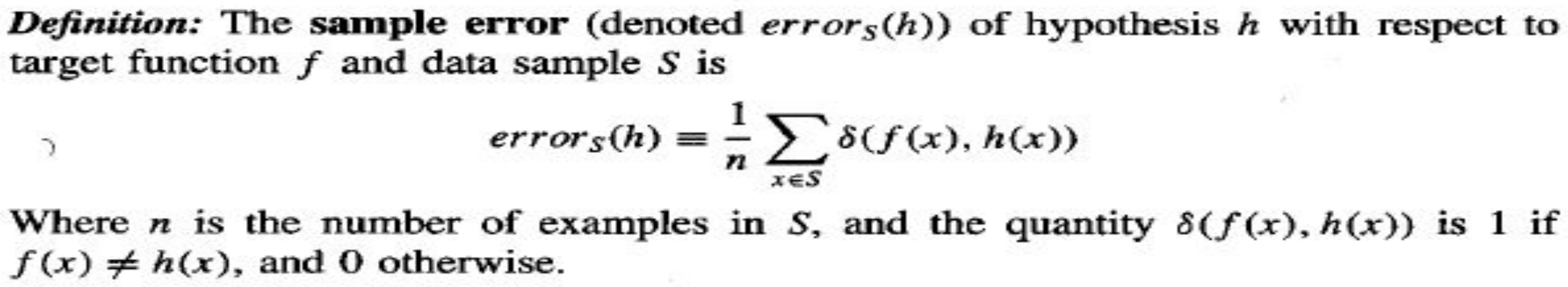
- Two things that are important while evaluating a learned hypothesis are:
  - Estimating the accuracy with which it will classify future instances.
  - Knowing the probable error in this accuracy estimate.
- The setting for the learning problems discussed here are:
  - There is some space of possible instances X over which various target functions may be defined.
  - Some unknown probability distribution D defines the probability of encountering each instance in X.
  - The Learning task is to learn the target function f by considering a space H of possible hypothesis.
  - Training examples are provided independently according to the distribution D.

Estimating Hypothesis Accuracy

- With the general setting discussed now we are interested in the following two questions:
  - Given a hypothesis h and a data sample containing n examples drawn at random according to the distribution D, what is the best estimate of the accuracy of h over future instances drawn from the same distribution?
  - What is the probable error in this accuracy estimate?

Sample Error and True Error

- We need to distinguish between two notions of accuracy or error:
  - One is the error rate of the hypothesis over the sample of data that is available (sample error).
  - The other is the error rate of the hypothesis over the entire unknown distribution D of examples (true error).
- The sample error of a hypothesis with respect to some sample S of instances drawn from X is the fraction of S that it misclassifies.



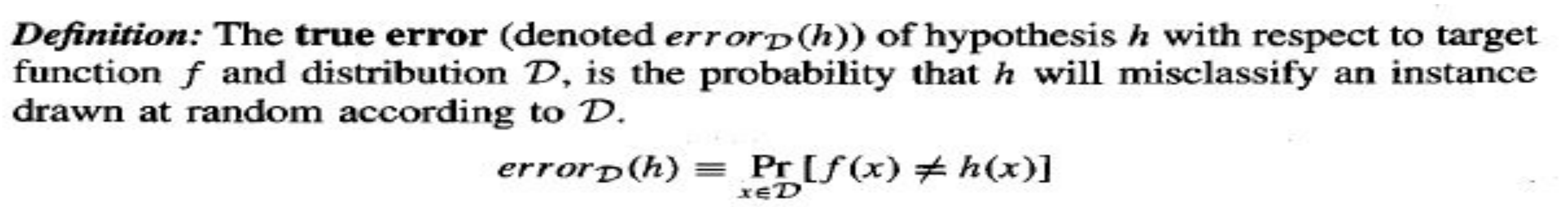
Definition: The **sample error** (denoted $error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where n is the number of examples in S, and the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

Sample Error and True Error

- The true error of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution D.



Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target function f and distribution $\mathcal{D}$, is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

- The most important question here is “How good an estimate of error_D(h) is provided by error_S(h)”.

Confidence intervals for Discrete Valued Hypothesis

- Here we find an answer to the question "How good an estimate of $\text{error}_D(h)$ is provided by $\text{error}_S(h)$" for the case which h is discrete valued hypothesis. Let us assume:
  - The sample *S* contains *n* examples drawn independent of one another, and independent of *h,* according to the probability distribution *D*
  - *n>=30*
  - Hypothesis *h* commits *r* errors over these *n* examples (i.e., $errors_S(h) = rln$).
- Under these conditions, statistical theory allows us to make the following assertions:
  - Given no other information, the most probable value of $\text{error}_D(h)$ is $\text{error}_S(h)$.
  - With approximately 95% probability the true error $\text{error}_D(h)$ lies in the interval

$$errors_S(h) \pm 1.96\sqrt{\frac{errors_S(h)(1 - errors_S(h))}{n}}$$

Confidence intervals for Discrete Valued Hypothesis

Confidence level $N\%$:	50%	68%	80%	90%	95%	98%	99%
Constant z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

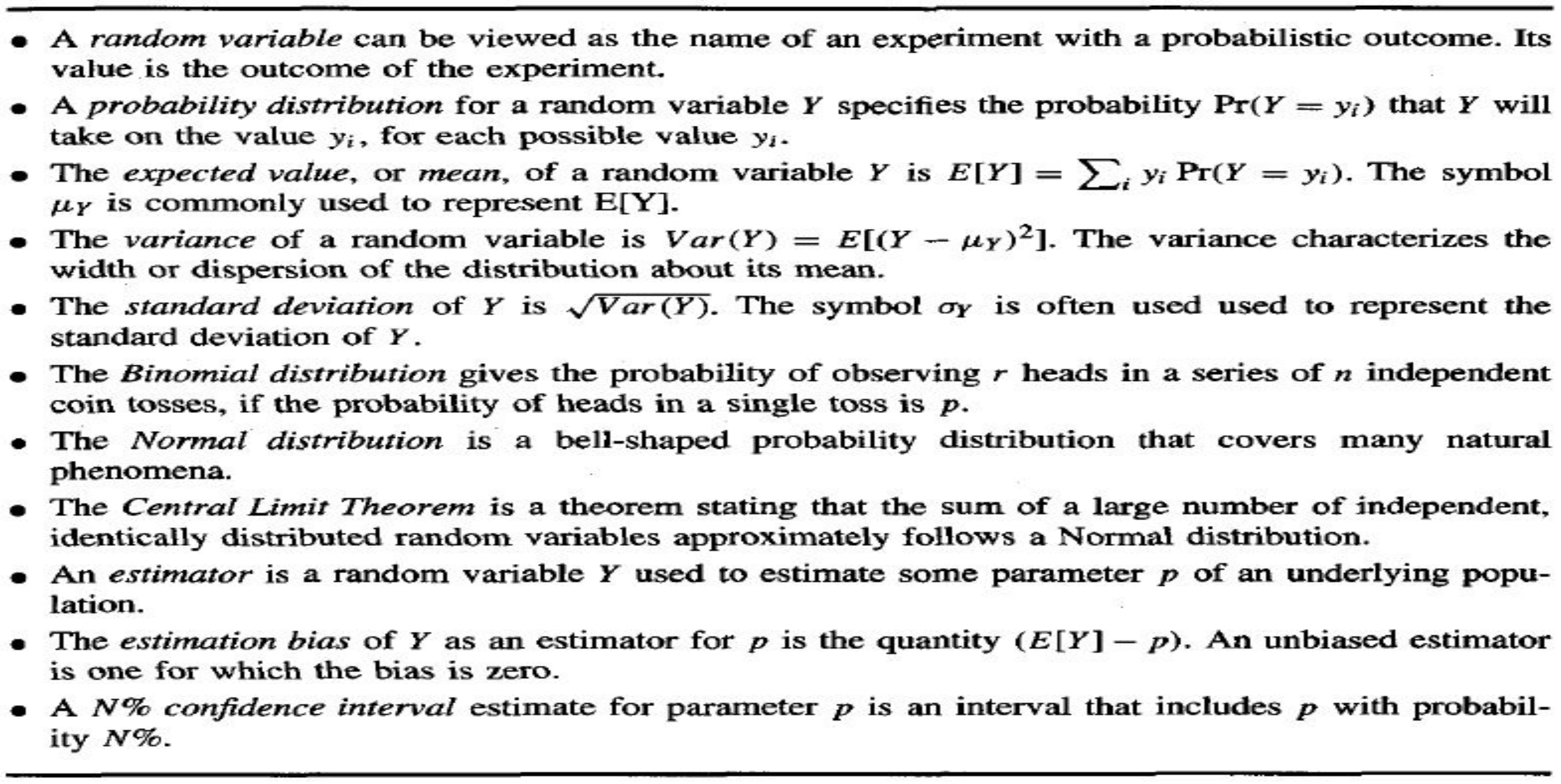
The above approximation works well when

$$n \ error_S(h)(1 - error_S(h)) \geq 5$$

Basics of Sampling Theory

- Error Estimation and Estimating Binomial Proportions
- The Binomial Distribution
- Mean and Variance
- Estimators, Bias and Variance
- Confidence Intervals
- Two-sided and One-sided Bounds

Basics of Sampling Theory- Basic Definitions and Facts from Statistics



- A *random variable* can be viewed as the name of an experiment with a probabilistic outcome. Its value is the outcome of the experiment.
- A *probability distribution* for a random variable Y specifies the probability $\Pr(Y = y_i)$ that Y will take on the value y_i, for each possible value y_i.
- The *expected value*, or *mean*, of a random variable Y is $E[Y] = \sum_i y_i \Pr(Y = y_i)$. The symbol μ_Y is commonly used to represent E[Y].
- The *variance* of a random variable is $Var(Y) = E[(Y - \mu_Y)^2]$. The variance characterizes the width or dispersion of the distribution about its mean.
- The *standard deviation* of Y is $\sqrt{Var(Y)}$. The symbol σ_Y is often used used to represent the standard deviation of Y.
- The *Binomial distribution* gives the probability of observing r heads in a series of n independent coin tosses, if the probability of heads in a single toss is p.
- The *Normal distribution* is a bell-shaped probability distribution that covers many natural phenomena.
- The *Central Limit Theorem* is a theorem stating that the sum of a large number of independent, identically distributed random variables approximately follows a Normal distribution.
- An *estimator* is a random variable Y used to estimate some parameter p of an underlying population.
- The *estimation bias* of Y as an estimator for p is the quantity $(E[Y] - p)$. An unbiased estimator is one for which the bias is zero.
- A *N% confidence interval* estimate for parameter p is an interval that includes p with probability $N\%$.

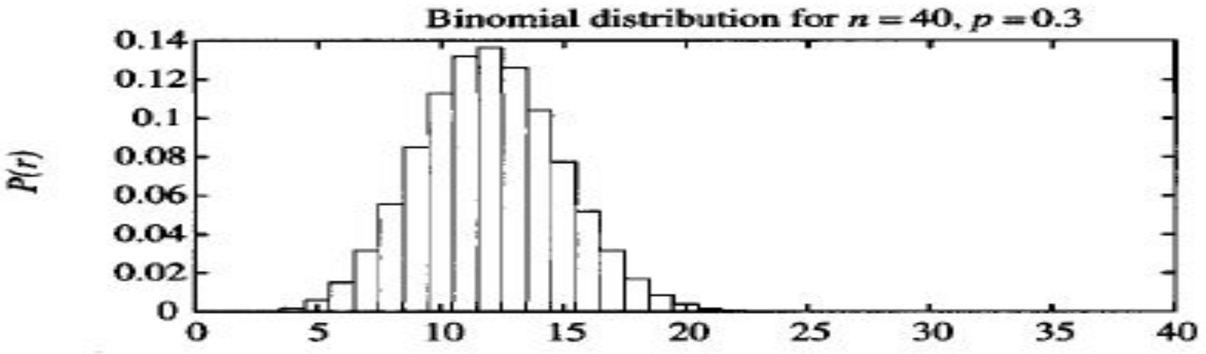
Error Estimation and Estimating Binomial Proportions

- How does the deviation between sample error and true error depend on the size of the data sample?
- The key to answering this question is to note that when we measure the sample error we are performing an experiment with a random outcome.
- We collect a random sample S of n independently drawn instances from the distribution D, and then measure the sample error $error_S(h)$.
- If the experiment is repeated number of times we say $error_{Si}(h)$ the outcome of the ith experiment is a random variable.

Error Estimation and Estimating Binomial Proportions

- Imagine we conducted k such experiments measuring the random variables $\text{error}_{S1}(h)$, $\text{error}_{S2}(h)$,…….$\text{error}_{Sk}(h)$.
- We then plot a histogram displaying the frequency with which we observed each possible error value.
- The histogram would approach a form of binomial distribution.

Error Estimation and Estimating Binomial Proportions- Binomial Distribution



A *Binomial distribution* gives the probability of observing r heads in a sample of n independent coin tosses, when the probability of heads on a single coin toss is p. It is defined by the probability function

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

If the random variable X follows a Binomial distribution, then:

- The probability $\Pr(X = r)$ that X will take on the value r is given by $P(r)$
- The expected, or mean value of X, $E[X]$, is

$$E[X] = np$$

- The variance of X, $Var(X)$, is

$$Var(X) = np(1-p)$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sqrt{np(1-p)}$$

For sufficiently large values of n the Binomial distribution is closely approximated by a Normal distribution (see Table 5.4) with the same mean and variance. Most statisticians recommend using the Normal approximation only when $np(1-p) \geq 5$.

Binomial Distribution

- Let us assume we are given a bent coin and asked to estimate the probability that the coin will turn up heads when tossed.
- The probability is p, the coin is tossed n times. We now record the number of times r that it turns up heads.
- A reasonable estimate of p is r/n.
- The binomial distribution describes for each possible value of r, the probability of observing exactly r heads given a sample of n independent tosses of a coin whose true probability of heads is p.
- Estimating p from a random sample of coin tosses is equivalent to estimating $error_D(h)$ from testing h on a random sample of instances.

Binomial Distribution

- The probability p that a single random coin toss will turn up heads corresponds to drawing a single random instance from D and determining whether it is misclassified by h.
- The number r of heads observed over a sample of n coin tosses correspond to the number of misclassifications observed over n randomly drawn instances.
- r/n corresponds to $\mathrm{error}_S(h)$.
- The problem of estimating p for coins is identical to the problem of estimating $\mathrm{error}_D(h)$ for hypotheses.

Binomial Distribution

- The general setting to which the binomial distribution applies is:
  - There is a base or underlying experiment whose outcome can be described by a random variable Y. The random variable can take two possible values.
  - The probability that *Y* = 1 on any single trial of the underlying experiment is given by some constant p, independent of the outcome of any other experiment. The probability that *Y* = 0 is therefore (1 - p). p is not known in advance, and the problem is to estimate it.
  - A series of n independent trials of the underlying experiment is performed producing the sequence of independent, identically distributed random variables $Y_1, Y_2, \ldots, Y_n$. Let *R* denote the number of trials for which $Y_i = 1$ in this series of n experiments.

$$R \equiv \sum_{i=1}^{n} Y_i$$

Binomial Distribution

- The probability that the random variable *R* will take on a specific value *r* (e.g., the probability of observing exactly *r* heads) is given by the Binomial distribution.

$$\Pr(R = r) = \frac{n!}{r!(n-r)!}\, p^r (1-p)^{n-r}$$

- The Binomial distribution characterizes the probability of observing *r* heads from n coin flip experiments, as well as the probability of observing *r* errors in a data sample containing n randomly drawn instances.

Mean and Variance

- Two properties of a random variable that are of interest are its expected value (mean) and its variance. The expected value is the average of the values taken on by repeatedly sampling the random variable.

Definition: Consider a random variable Y that takes on the possible values $y_1, \ldots y_n$. The **expected value** of Y, $E[Y]$, is

$$E[Y] = \sum_{i=1}^{n} y_i \Pr(Y = y_i) \tag{5.3}$$

- In case the random variable Y is governed by a binomial distribution then it can be shown that

$$E[Y] = np$$

Where p and n are the parameters of binomial distribution.

Mean and Variance

- A second property, the variance, captures the "width or "spread" of the probability distribution; that is, it captures how far the random variable is expected to vary from its mean value.

Definition: The **variance** of a random variable Y, $Var[Y]$, is

$$Var[Y] \equiv E[(Y - E[Y])^2]$$

- The square root of variance is called the standard deviation of Y, denoted σ_Y

Definition: The **standard deviation** of a random variable Y, σ_Y, is

$$\sigma_Y \equiv \sqrt{E[(Y - E[Y])^2]}$$

Mean and Variance

- In case the random variable Y is governed by a Binomial distribution, then the variance and standard deviation are given by

$$Var[Y] = np(1-p)$$

$$\sigma_Y = \sqrt{np(1-p)}$$

Estimators, Bias and Variance

- Now we return back to our primary question "What is the likely difference between $error_S(h)$ and the true error $error_D(h)$".
- According to the definition of binomial distribution:

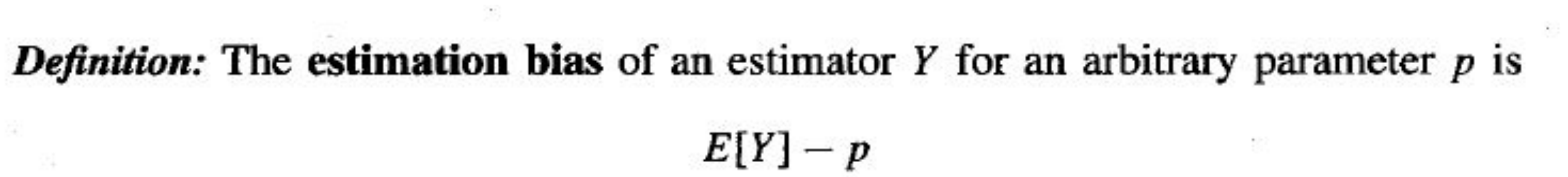
$$error_S(h)=r/n$$

$$error_D(h)=p$$

- where *n* is the number of instances in the sample S, *r* is the number of instances from S misclassified by *h,* and p is the probability of misclassifying a single instance drawn from *D.*

Estimators, Bias and Variance

- Statisticians call *error$_S$ (h)* an *estimator* for the true error *error$_D$(h).*
- We define the *estimation bias* to be the difference between the expected value of the estimator and the true value of the parameter.



Definition: The **estimation bias** of an estimator Y for an arbitrary parameter p is

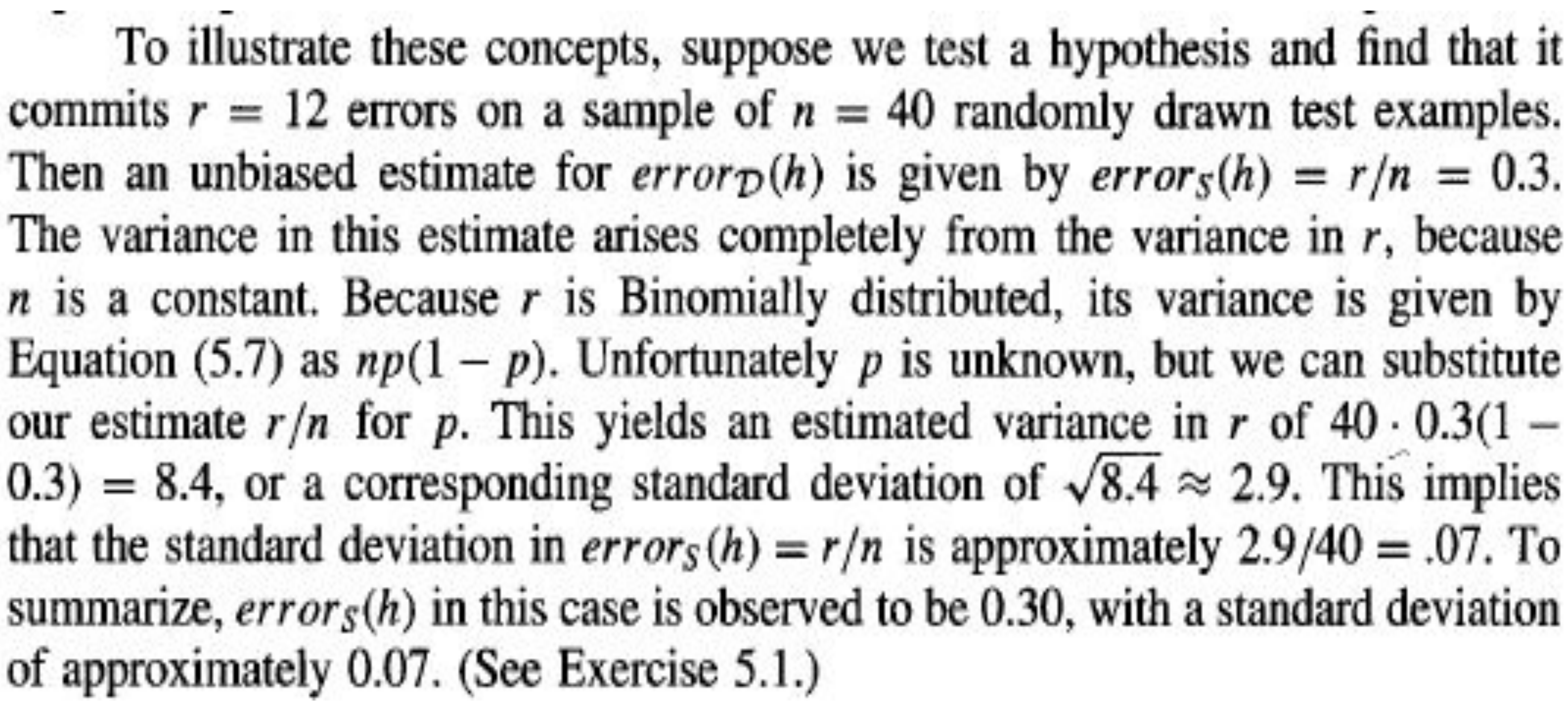
$$E[Y] - p$$

- If the estimation bias is zero, we say that Y is an *unbiased estimator* for *p.*
- In the case of binomial distribution error$_S$(h) is an unbiased estimator for error$_D$(h) because for binomial distribution the expected value of r is np and given n constant expected value of r/n is p.

Estimators, Bias and Variance

- In order for $error_S(h)$ to give an unbiased estimate of $error_D(h)$. The hypothesis h and sample S must be chosen independently.
- The notion of estimation bias should not be confused with the inductive bias of a learner.
- Another important property of an estimator is its variance.
- Given a choice among alternative unbiased estimators, it makes sense to choose the one with least variance.
- By our definition of variance, this choice will yield the smallest expected squared error between the estimate and the true value of the parameter.

Estimators, Bias and Variance



To illustrate these concepts, suppose we test a hypothesis and find that it commits $r = 12$ errors on a sample of $n = 40$ randomly drawn test examples. Then an unbiased estimate for $error_{\mathcal{D}}(h)$ is given by $error_S(h) = r/n = 0.3$. The variance in this estimate arises completely from the variance in r, because n is a constant. Because r is Binomially distributed, its variance is given by Equation (5.7) as $np(1-p)$. Unfortunately p is unknown, but we can substitute our estimate r/n for p. This yields an estimated variance in r of $40 \cdot 0.3(1 - 0.3) = 8.4$, or a corresponding standard deviation of $\sqrt{8.4} \approx 2.9$. This implies that the standard deviation in $error_S(h) = r/n$ is approximately $2.9/40 = .07$. To summarize, $error_S(h)$ in this case is observed to be 0.30, with a standard deviation of approximately 0.07. (See Exercise 5.1.)

Estimators, Bias and Variance

- Given r errors in a sample of n independently drawn test examples the standard deviation for $\text{error}_S(h)$ is given by

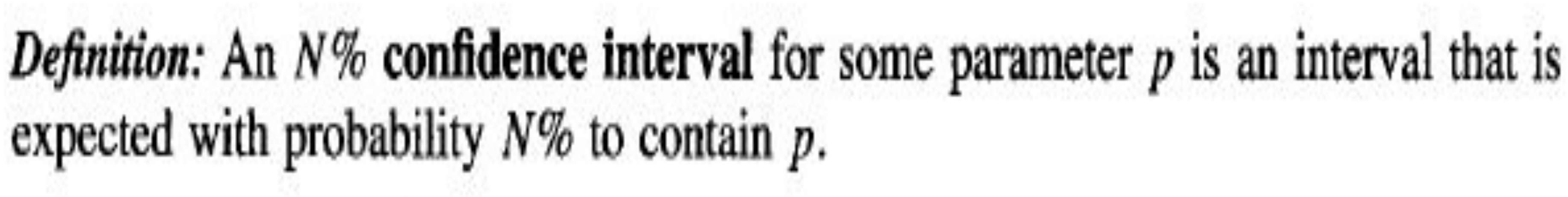
$$\sigma_{error_S(h)} = \frac{\sigma_r}{n} = \sqrt{\frac{p(1-p)}{n}}$$

which can be approximated by substituting $r/n = error_S(h)$ for p

$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

Confidence Intervals

- One common way to describe the uncertainty associated with an estimate is to give an interval within which the true value is expected to fall, along with the probability with which it is expected to fall into this interval. Such estimates are called ***confidence interval*** estimates.



Definition: An $N\%$ **confidence interval** for some parameter p is an interval that is expected with probability $N\%$ to contain p.

Confidence Intervals

- Here we need to derive the confidence intervals for $error_D(h)$.
- As we know binomial distribution governs the estimator $error_S(h)$. The mean value of this distribution is $error_D(h)$. The standard deviation is:

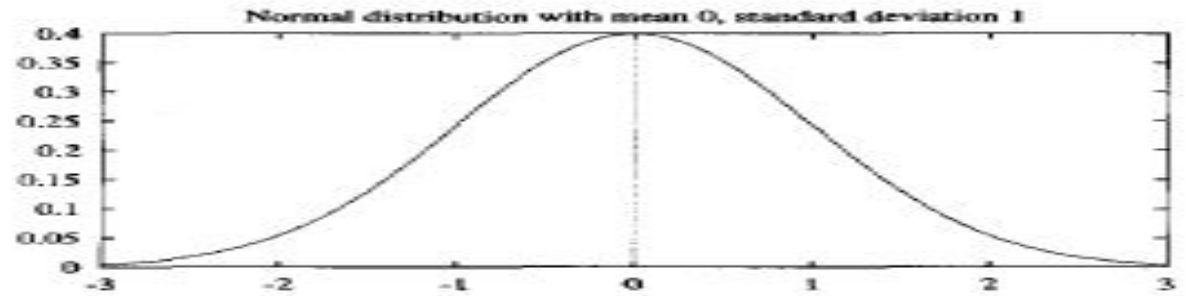
$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

- To derive a 95% confidence interval we need to find the interval centered around the mean value $error_D(h)$, which is wide enough to contain 95% of the total probability under this distribution.
- This provides an interval surrounding $error_D(h)$ into which $error_S(h)$ must fall 95% of the time.

Confidence Intervals

- Now given a size of N we need to find the size of the interval N% of the probability mass.
- This calculation is very difficult for binomial distribution but for a sufficiently large sample binomial distribution can be closely approximated by the normal distribution.
- It is a bell shaped distribution fully specified by its mean μ and σ standard deviation.
- For a larger n the binomial distribution is very closely approximated by a normal distribution with the same mean and variance.

Confidence Intervals



A Normal distribution (also called a Gaussian distribution) is a bell-shaped distribution defined by the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

A Normal distribution is fully determined by two parameters in the above formula: μ and σ.

If the random variable X follows a normal distribution, then:

- The probability that X will fall into the interval (a, b) is given by

$$\int_a^b p(x)dx$$

- The expected, or mean value of X, $E[X]$, is

$$E[X] = \mu$$

- The variance of X, $Var(X)$, is

$$Var(X) = \sigma^2$$

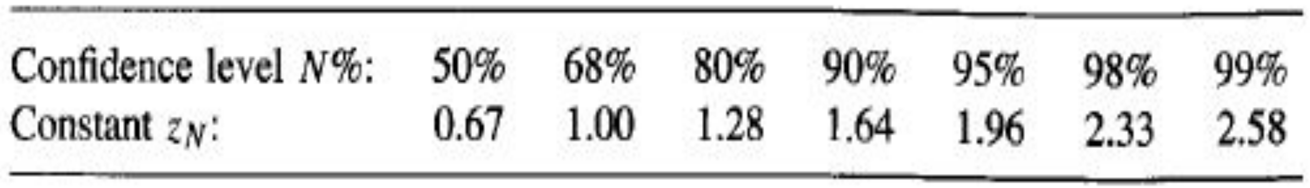
- The standard deviation of X, σ_X, is

$$\sigma_X = \sigma$$

The Central Limit Theorem (Section 5.4.1) states that the sum of a large number of independent, identically distributed random variables follows a distribution that is approximately Normal.

Confidence Intervals

- We have tables specifying the size of the interval bout the mean that contains N% of the probability mass under the normal distribution.



Confidence level $N\%$:	50%	68%	80%	90%	95%	98%	99%
Constant z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

- The constant ZN defines the width of the smallest interval about the mean that includes N% of the total probability mass under the bell shaped normal distribution. It actually gives half the width of the interval.

Confidence Intervals

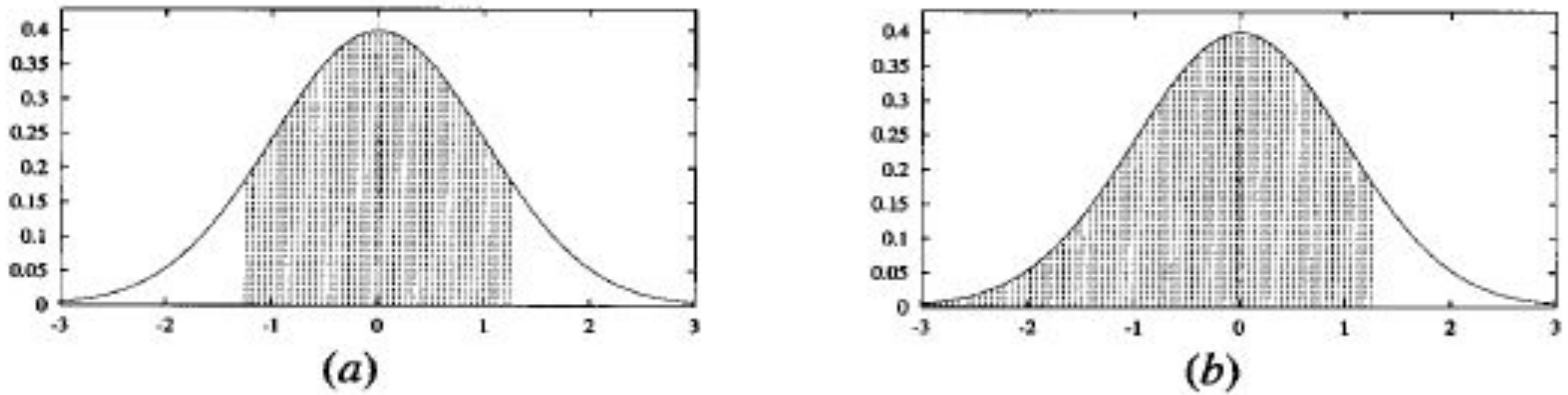


FIGURE 5.1
A Normal distribution with mean 0, standard deviation 1. (*a*) With 80% confidence, the value of the random variable will lie in the two-sided interval $[-1.28, 1.28]$. Note $z_{.80} = 1.28$. With 10% confidence it will lie to the right of this interval, and with 10% confidence it will lie to the left. (*b*) With 90% confidence, it will lie in the one-sided interval $[-\infty, 1.28]$.

Confidence Intervals

- If a random variable Y obeys a normal distribution with mean μ and standard deviation σ the measured value y of Y will fall into the following interval N% of the time

$$\mu \pm z_N \sigma$$

- The mean μ will fall into the following interval N% of the time

$$y \pm z_N \sigma$$

- Substituting the standard deviation for errorS(h) in the above equation we get the expression for N% confidence intervals for discrete valued hypothesis

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

Confidence Intervals

- Two approximations were involved in deriving the previous expression
  - In estimating the standard deviation σ of errorS(h) we have approximated errorD(h) by errorS(h).
  - The binomial distribution has been approximated by the normal distribution.

Two-Sided and One-Sided Bounds

- The confidence interval discussed so far is two-sided bound.
- In some cases we will be interested in one sided bound like “What is the probability that $error_D(h)$ is at most U?”
- The previous calculation can be easily modified to find a one sided bound.
- As we know normal distribution is symmetric about its mean, the two sided confidence interval can be converted to one sided with twice the confidence.
- A 100(1-α)% confidence interval with lower bound L and upper bound U implies a 100(1- α/2)% confidence with upper bound U.

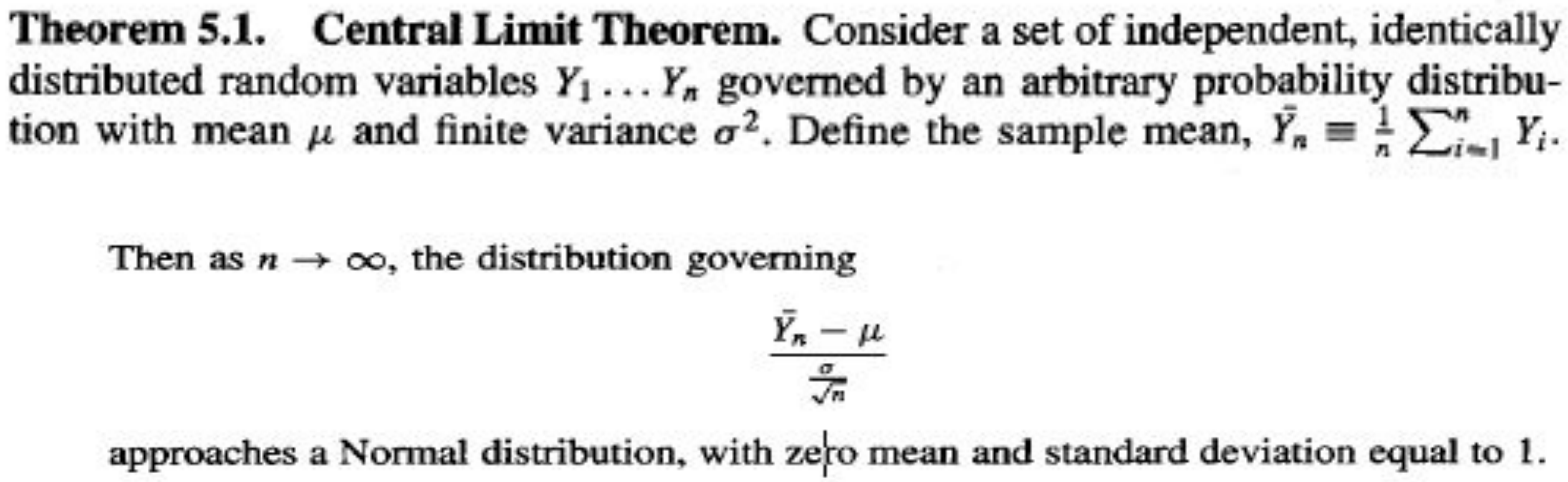
A General Approach for Deriving Confidence Intervals

- The previous section described in detail how to derive confidence interval estimates for one particular case: estimating $error_D(h)$ for a discrete-valued hypothesis *h,* based on a sample of *n* independently drawn instances.
- We can see this as a problem of estimating the mean (expected value) of a population based on the mean of a randomly drawn sample of size *n.*

A General Approach for Deriving Confidence Intervals

- The general process includes the following steps:
  - Identify the underlying population parameter p to be estimated, for example, $error_D(h)$.
  - Define the estimator Y (e.g., $error_S(h)$). It is desirable to choose a minimum variance, unbiased estimator.
  - Determine the probability distribution D_Y that governs the estimator Y, including its mean and variance.
  - Determine the *N%* confidence interval by finding thresholds L and U such that *N%* of the mass in the probability distribution D_Y falls between L and U.

Central Limit Theorem



Theorem 5.1. Central Limit Theorem. Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$ governed by an arbitrary probability distribution with mean μ and finite variance σ^2. Define the sample mean, $\bar{Y}_n \equiv \frac{1}{n}\sum_{i=1}^{n} Y_i$.

Then as $n \to \infty$, the distribution governing

$$\frac{\bar{Y}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$$

approaches a Normal distribution, with zero mean and standard deviation equal to 1.

Difference in Error of Two Hypothesis

- Let us consider the case where we have two hypothesis h_1 and h_2 for some discrete valued target function.
- Hypothesis h_1 has been tested on sample S_1 containing n_1 randomly drawn examples.
- Hypothesis h_2 has been tested on sample S_2 containing n_2 randomly drawn examples.
- The difference d between the true error of these two hypothesis is

$$d \equiv error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$$

Difference in Error of Two Hypothesis

- It is decided that d is the parameter to be estimated. Now we define an estimator. The estimator in this case will be the difference between the sample errors denoted by $\hat{d}$

$$\hat{d} \equiv error_{S_1}(h_1) - error_{S_2}(h_2)$$

- $\hat{d}$ gives an unbiased estimate of d that is $E[\hat{d}\ ] = d.$
- What is the probability distribution governing the random variable $\hat{d}$?
- The difference of two normal distributions will follow normal distribution $\hat{d}$ will also follow normal distribution with mean d.

Difference in Error of Two Hypothesis

- The variance of this distribution is the sum of the variances of $\text{error}_{S1}(h1)$ and $\text{error}_{S2}(h2)$.

$$\sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)(1-error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2}$$

- It is easy to derive the confidence intervals that characterize the likely error in employing $\hat{d}$ to estimate d.
- For a random variable $\hat{d}$ obeying a normal distribution with mean d and variance σ^2 the N% confidence interval for d is

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1-error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2}}$$

Difference in Error of Two Hypothesis

- The same interval can be used though it is overly conservative when the hypotheses h1 and h2 are used on the same sample S. Then

$$\hat{d} \equiv error_S(h_1) - error_S(h_2)$$

Hypothesis Testing

- Suppose we are interested in the question "What is the probability that $error_D(h_1) > error_D(h_2)$?"
- Let sample errors for h1 and h2 using two independent samples S1 and S2 of size 100. Let errorS1(h1)=.30 and errorS2(h2)=.20, the observed difference $\hat{d}$ =.10.
- This random variation can occur even when $error_D(h_1) <= error_D(h_2)$.
- Finally the question can be modified as what is the probability that d>0 given $\hat{d}$ =.10.
- The probability Pr(d>0) is equals the probability tha $\hat{d}$ falls into one sided interval

$$\hat{d} < \mu_{\hat{d}} + .10.$$

Hypothesis Testing

- Substituting the given values in the equation for variance we get

$$\sigma_{\hat{d}} \approx .061$$

- Substituting the above value we can redefine the interval as:

$$\hat{d} < \mu_{\hat{d}} + 1.64\sigma_{\hat{d}}$$

- Consulting the Z table we find that 1.64 standard deviation about the mean corresponds to a two sided interval with confidence 90%.
- The one sided interval will be associated with a confidence of 95%
- We can finally say that "errror$_D$(h$_1$)>error$_D$(h$_2$)" with 95% confidence.

Comparing Learning Algorithms

- Paired t tests
- Practical Considerations

Comparing Learning Algorithms

- What is an appropriate test for comparing learning algorithms L_A and L_B?
- How can we determine whether an observed difference between the algorithms is statistically significant?
- Suppose we wish to determine which of LA and LB is the better learning method on average for learning some particular target function f.
- Here we wish to estimate the expected values of the difference in their errors.

$$\underset{S \subset \mathcal{D}}{E}\left[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))\right]$$

Where L(S) denotes the hypothesis output by learning method L when given sample data S of training data.

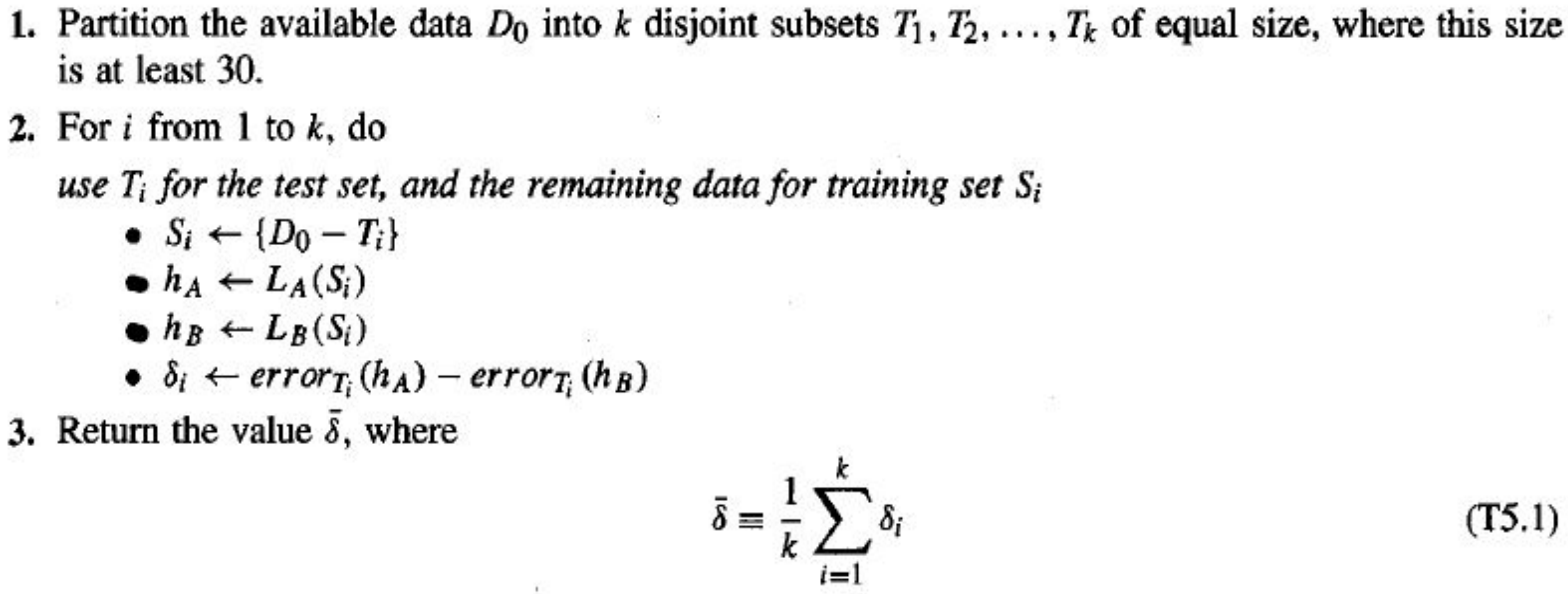
Comparing Learning Algorithms

- When we have limited sample D_0 we divide it into training set S_0 and test set T_0.
- We now measure the quantity

$$error_{T_0}(L_A(S_0)) - error_{T_0}(L_B(S_0))$$

- The above is the estimator.
- One way to improve on the estimator is to repeatedly partition the data D_0 into disjoint training and test sets and to take the mean of the experiments.

Comparing Learning Algorithms



1. Partition the available data D_0 into k disjoint subsets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.
2. For i from 1 to k, do

   use T_i for the test set, and the remaining data for training set S_i

   - $S_i \leftarrow \{D_0 - T_i\}$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$
3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k}\sum_{i=1}^{k} \delta_i \qquad \text{(T5.1)}$$

Comparing Learning Algorithms

- The quantity $\bar{\delta}$ returned by the previous procedure can be takes as an estimate of the desired quantity specified earlier.
- We can view $\bar{\delta}$ as an estimate of the quantity

$$\underset{S \subset D_0}{E}[error_{\mathcal{D}}(L_A(S)) - error_{\mathcal{D}}(L_B(S))]$$

where S represents a random sample of size $\frac{k-1}{k}|D_0|$ drawn uniformly from D_0.

- D_0 is the available data.

Comparing Learning Algorithms

- The approximate N% confidence interval for estimating the quantity in the above equation using $\bar{\delta}$ is given by

$$\bar{\delta} \pm t_{N,k-1}\ s_{\bar{\delta}}$$

where $t_{N,k-1}$ is a constant which plays the same role as Z_N and $s_{\bar{\delta}}$ an estimate of the standard deviation of the distribution governing . In $\bar{\delta}$ articular is def $s_{\bar{\delta}}$ ed as

$$s_{\bar{\delta}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (\delta_i - \bar{\delta})^2}$$

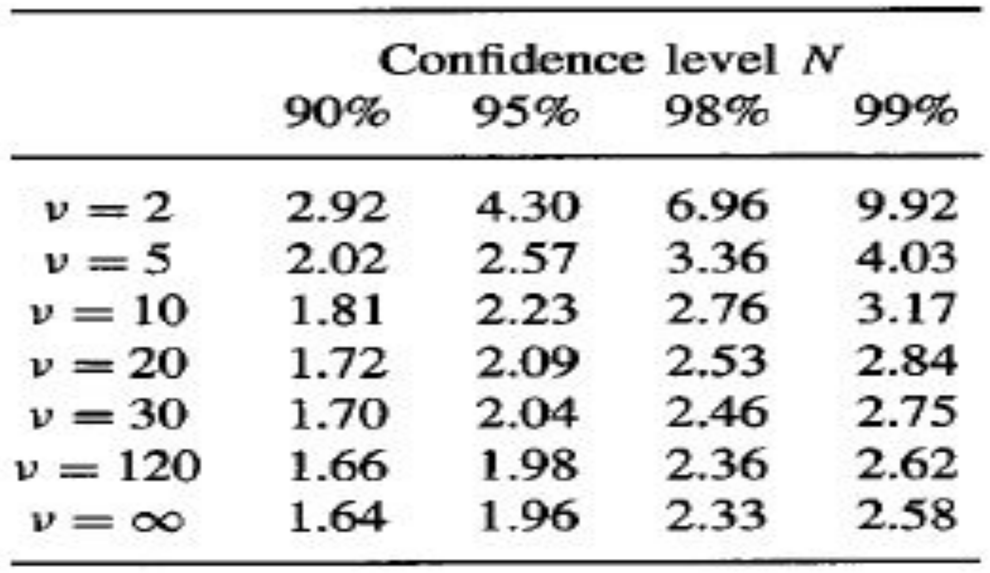
- The constant tN,k-1 has two subscripts. The first specifies the desired confidence level and the second called the number of degrees of freedom denoted by v is the number of independent random events that go into producing the value for the random variable $\bar{\delta}$ Here the value is k-1.

Comparing Learning Algorithms

- The procedure for comparing two learning methods involves testing the two learned hypothesis on identical test sets.
- Tests where the hypotheses are evaluated over identical samples are called paired tests.
- Paired tests produce tighter confidence intervals because any difference in observed errors in a paired test are due to differences between the hypotheses.
- When the hypotheses are tested on separate data samples differences in the two sample errors might be due to the makeup of the two samples.

Comparing Learning Algorithms

- The values for the parameter t are given in the following table.



	Confidence level N			
	90%	95%	98%	99%
$\nu = 2$	2.92	4.30	6.96	9.92
$\nu = 5$	2.02	2.57	3.36	4.03
$\nu = 10$	1.81	2.23	2.76	3.17
$\nu = 20$	1.72	2.09	2.53	2.84
$\nu = 30$	1.70	2.04	2.46	2.75
$\nu = 120$	1.66	1.98	2.36	2.62
$\nu = \infty$	1.64	1.96	2.33	2.58

TABLE 5.6
Values of $t_{N,\nu}$ for two-sided confidence intervals. As $\nu \to \infty$, $t_{N,\nu}$ approaches z_N.

Paired t Tests

- Here we discuss about the statistical justification for the procedure discussed for getting the confidence interval.
- Let us consider the following estimation problem:
  - We are given the observed values of a set of independent, identically distributed random variable $Y_1, Y_2 \ldots Y_{k.}$
  - We wish to estimate the mean µ of the probability distribution governing these Yi.
  - The estimator we will use is the sample mean $\bar{Y}$

$$\bar{Y} \equiv \frac{1}{k} \sum_{i=1}^{k} Y_i$$

Paired t Tests

- This problem is the problem of estimating the distribution mean μ based on the sample mean $\bar{Y}$.
- It is similar to the problem discussed earlier using $error_S(h)$ to estimate $error_D(h)$. In that problem the Y_i are 1 or 0.
- The t test discussed here applies to a special case of this problem, the case in which the individual Y_i follow a normal distribution.

Paired t Tests

- Instead of having a fixed sample of data D_0 we request new training examples drawn according to the underlying instance distribution.
- The procedure for calculation of $\bar{\delta}$ is modified so that on each iteration through the loop it generates a new random training set S_i and new random test set T_i by drawing from this underlying instance distribution instead of drawing from the fixed sample D_0.
- This method fits the form of the above estimation problem.
- The mean μ of their distribution corresponds to the expected difference in error between the two learning methods.
- The sample mean $\bar{Y}$ is the quantity $\bar{\delta}$ computed after the changes.
- Now the question is "how good an estimate of μ is provided by $\bar{Y}$"

Paired t Tests

- Now the equation for the calculation of μ becomes:

$$\mu = \bar{Y} \pm t_{N,k-1}\ s_{\bar{Y}}$$

Where $s_{\bar{Y}}$ is the estimated standard deviation of the sample mean

$$s_{\bar{Y}} \equiv \sqrt{\frac{1}{k(k-1)} \sum_{i=1}^{k} (Y_i - \bar{Y})^2}$$

- The constant tN,k-1 is a constant similar to ZN. It characterizes the area under a probability distribution known as the t distribution.
- The t distribution approaches normal distribution as k approaches infinity.

Practical Considerations

- The method used for the calculation error in the algorithm described is called k-fold approach.
- Another approach is to select a test set of at least 30 examples from D_0 and use the remaining for training. This process can be repeated as many times as desired.
- The k-fold method is limited by the number of examples.
- The randomized method has the disadvantage that the test sets no longer qualify as independently drawn from the underlying distribution D.