Artificial Neural Networks

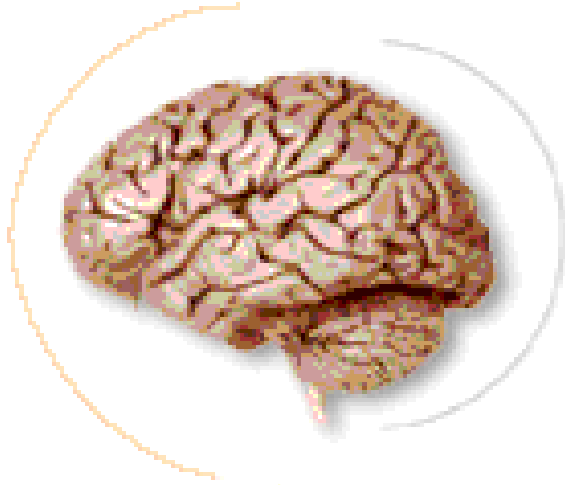# Contents

Introduction

Perceptron and Gradient Descent Algorithm

Multilayer Neural Networks
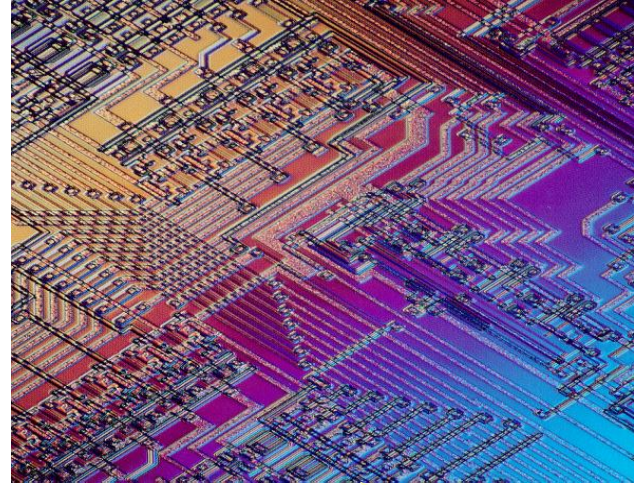
Designing an ANN for Face Recognition Application

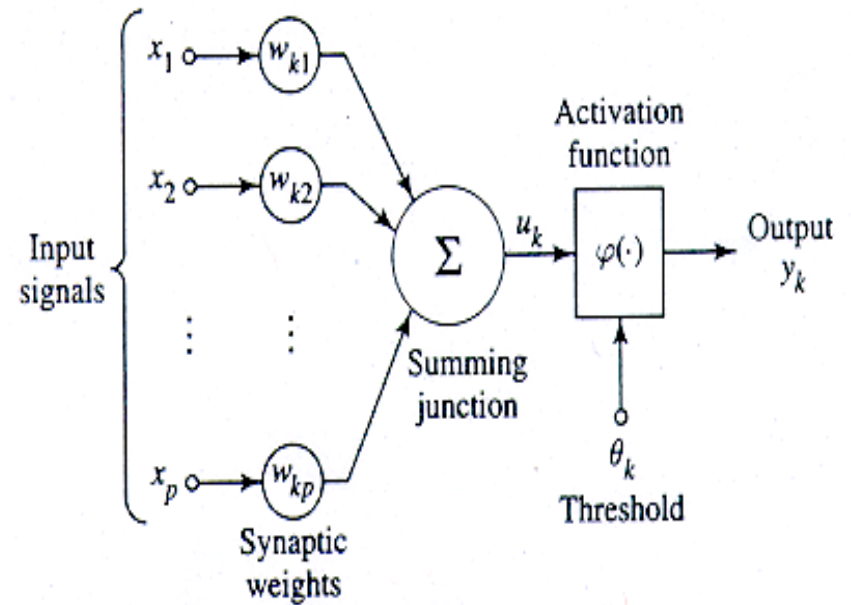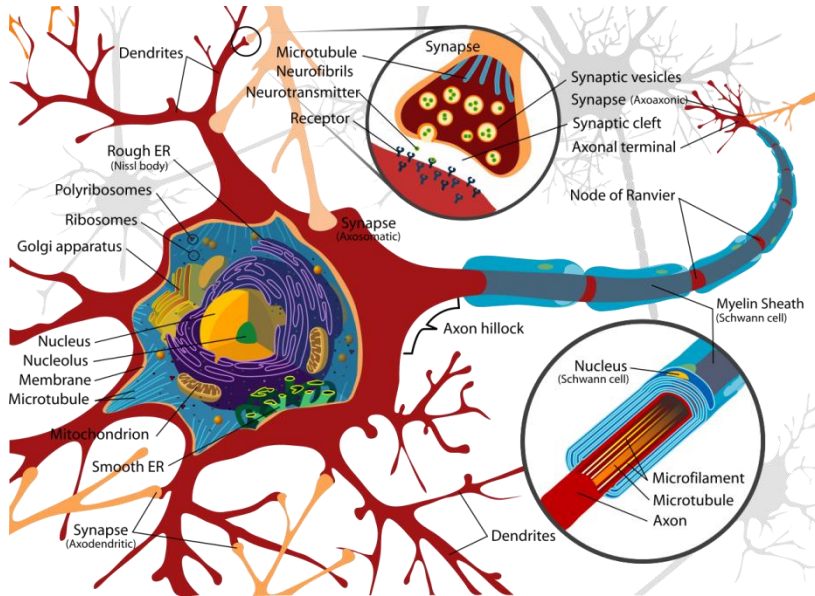# Introduction

The Brain vs. Computer

1. 10 billion neurons
2. 60 trillion synapses
3. Distributed processing
4. Nonlinear processing
5. Parallel processing

1. Faster than neuron (10^{-9} sec)
   cf. neuron: 10^{-3} sec
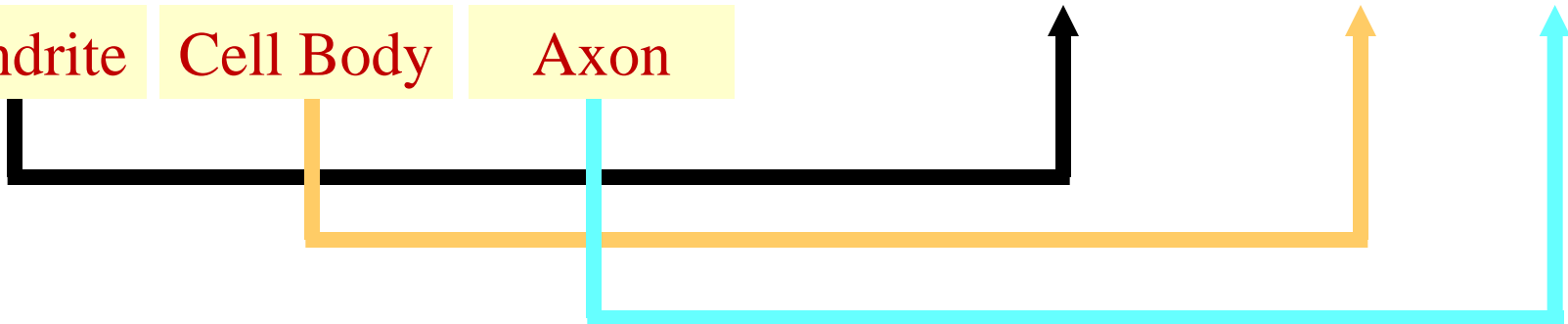3. Central processing
4. Arithmetic operation (linearity)
5. Sequential processing

From Biological Neuron to Artificial Neuron

Dendrite   Cell Body   Axon

From Biology to Artificial Neural Networks
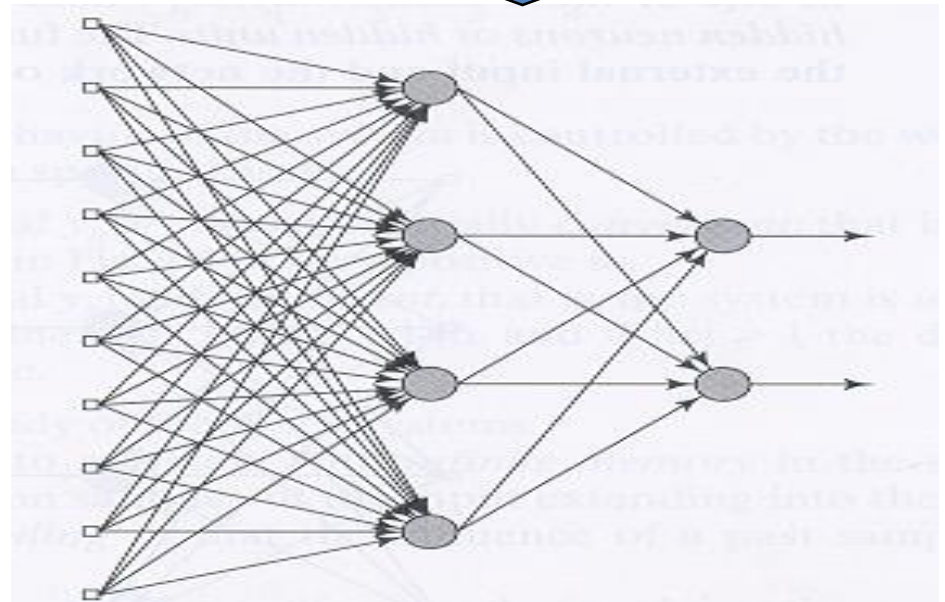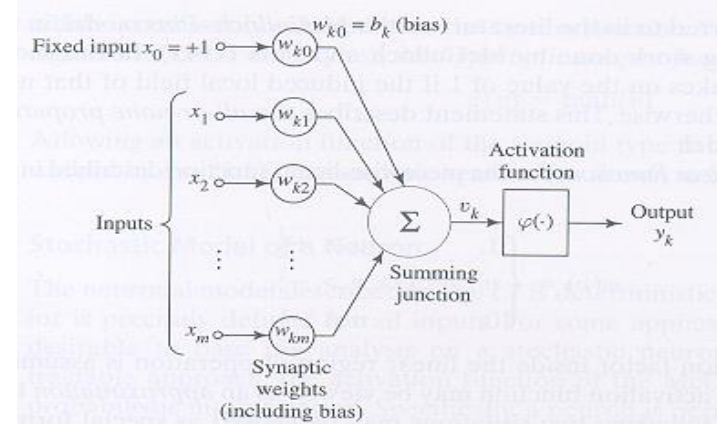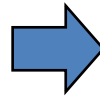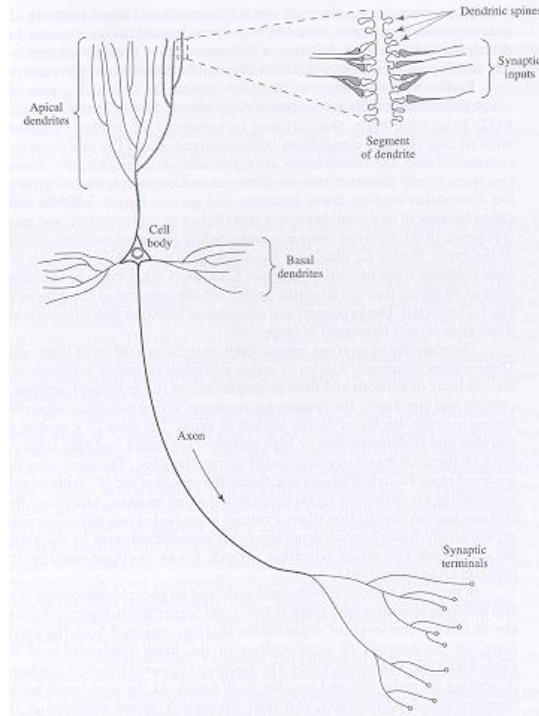
Properties of Artificial Neural Networks

- A network of artificial

Input layer · First hidden layer · Second hidden layer · Output layer

\<Multilayer Perceptron Network\>
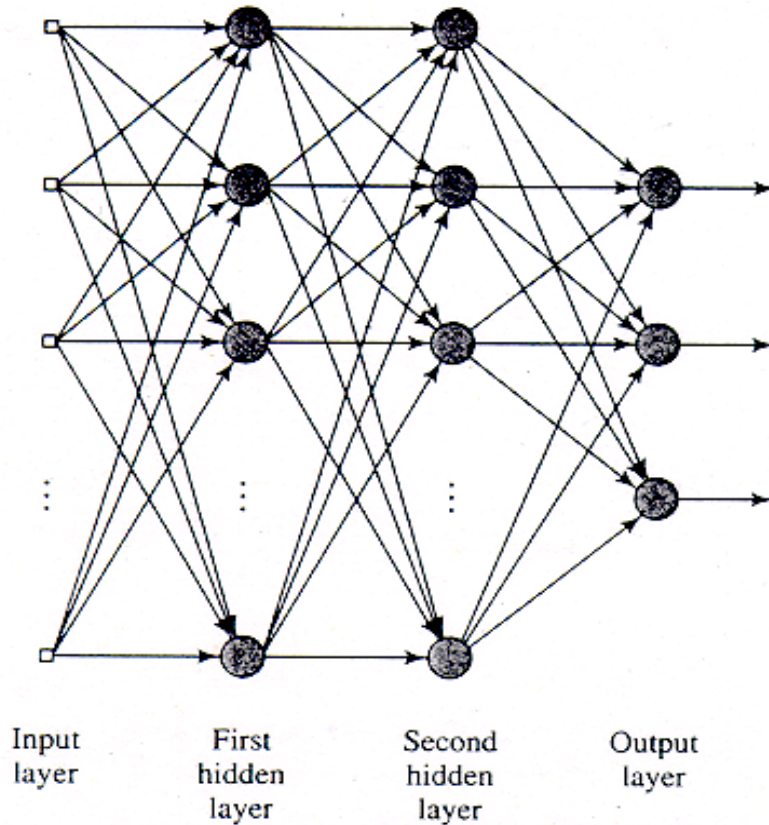
Characteristics

- Nonlinear I/O mapping
- Adaptivity
- Generalization ability
- Fault-tolerance (graceful degradation)
- Biological analogy

Types of ANNs

❖ Single Layer Perceptron

❖ Multilayer Perceptrons (MLPs)

❖ Radial-Basis Function Networks (RBFs)

❖ Hopfield Network

❖ Boltzmann Machine

❖ Self-Organization Map (SOM)

❖ Modular Networks (Committee Machines)

Architectures of Networks

<Multilayer Perceptron Network>

<Hopfield Network>
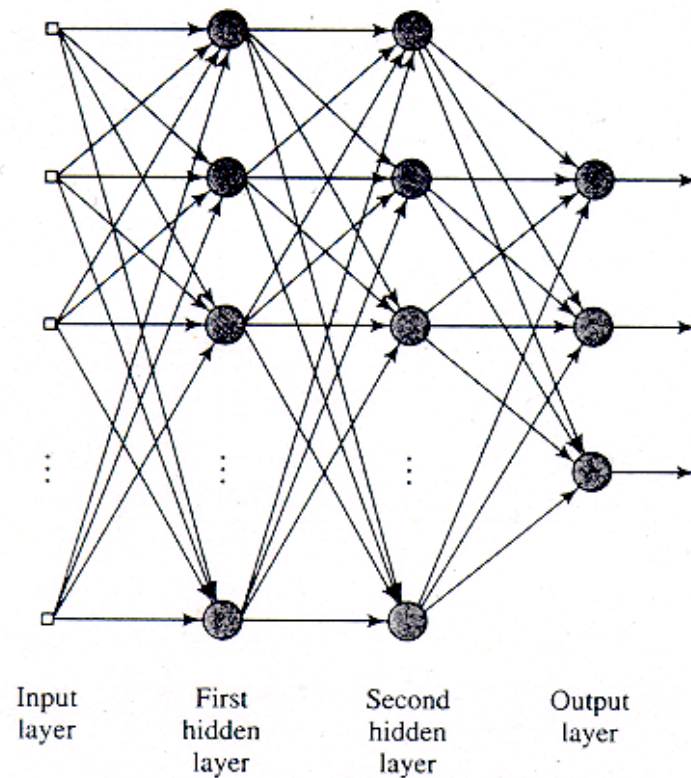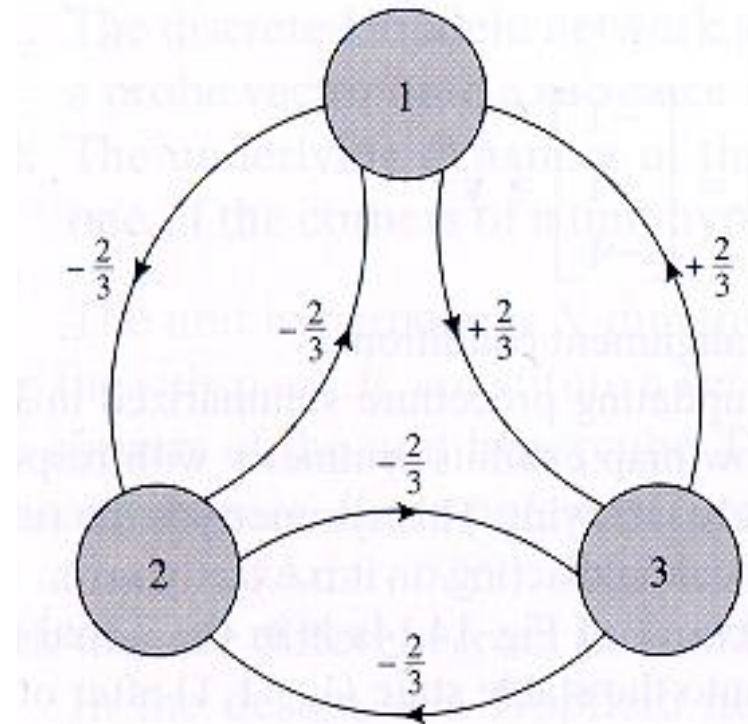
Features of Artificial Neural Networks

❖ Records (examples) need to be represented as a (possibly large) set of tuples of <attribute, value>

❖ The output values can be represented as a discrete value, a real value, or a vector of values

❖ Tolerant to noise in input data

❖ Time factor

   ❖ It takes long time for training

   ❖ Once trained, an ANN produces output values (predictions) fast

❖ It is hard for human to interpret the process of prediction by ANN

Example of Applications

- NETtalk [Sejnowski]
  - Inputs: English text
  - Output: Spoken phonemes

- Phoneme recognition [Waibel]
  - Inputs: wave form features
  - Outputs: b, c, d,...

- Robot control [Pomerleau]
  - Inputs: perceived features
  - Outputs: steering control

Application:
Autonomous Land Vehicle (ALV)

- NN learns to steer an autonomous vehicle.
- 960 input units, 4 hidden units, 30 output units
- Driving at speeds up to 70 miles per hour
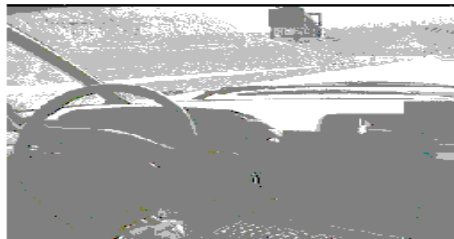
ALVINN System

Image of a forward - mounted camera
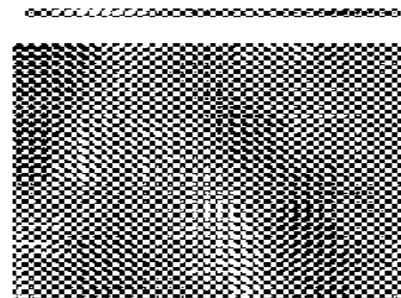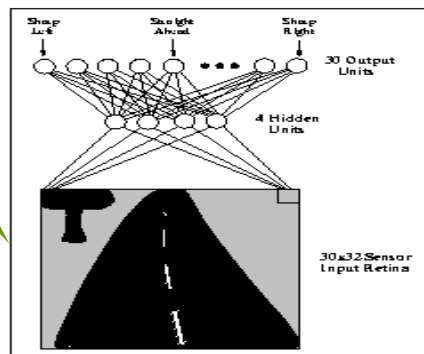
Weight values for one of the hidden units

Application:
Error Correction by a Hopfield Network

corrupted input data

original target data
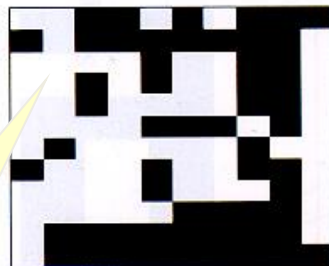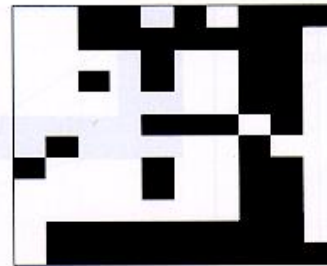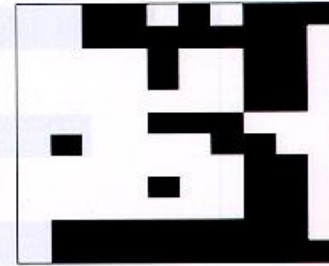
Corrected data after 20 iterations

Corrected data after 10 iterations
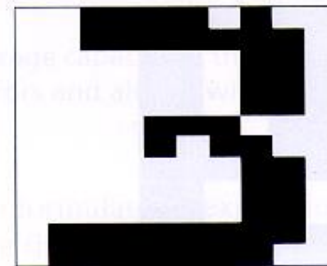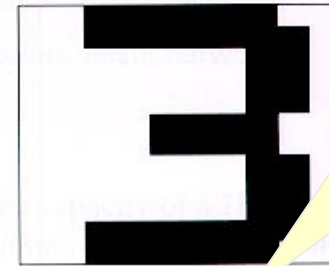
Fully corrected data after 35 iterations

Perceptron
and
Gradient Descent Algorithm

Architecture of A Perceptron
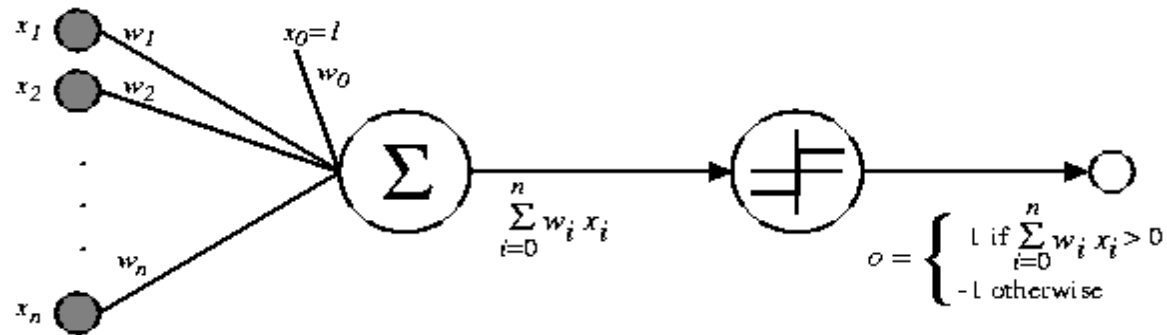
$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

- Input: a vector of real values
- Output: 1 or -1 (binary)
- Activation function: threshold function

NOTE: Perceptron is also called as a TLU (Threshold Logic Unit)

Hypothesis Space of Perceptrons

- Free parameters: weights (and thresholds)

- Learning: choosing values for the weights

- Hypotheses space of perceptron learning
$$H = \{\vec{w} \mid \vec{w} \in \Re^{(n+1)}\}$$

  - n: dimension of the input vector

  - Linear function

Perceptron and Decision Hyperplane

- A perceptron represents a 'hyperplane' decision surface in the n-dimensional space of instances (*i.e.* points).
- The perceptron outputs 1 for instances lying on one side of the hyperplane and outputs -1 for instances lying on the other side.
- Equation for the decision hyperplane: **wx** = 0.
- Some sets of positive and negative examples cannot be separated by any hyperplane

- A perceptron can not learn a linearly nonseparable problem.

Linearly Separable *v.s.* Linearly Nonseparable
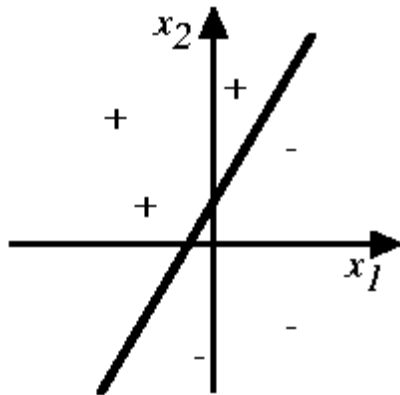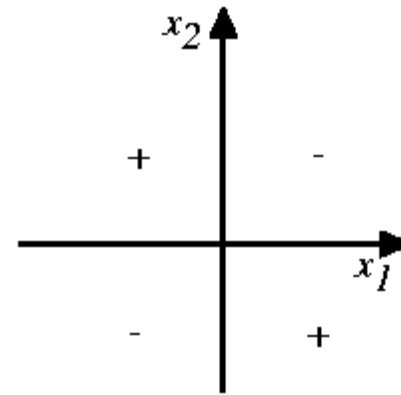
(a) Decision surface for a *linearly separable* set of examples (correctly classified by a straight line)

(b) A set of training examples that is *not linearly separable*.

Representational Power of Perceptrons

- A single perceptron can be used to represent many boolean functions.
  - AND function: $w_0 = -0.8$, $w_1 = w_2 = 0.5$
  - OR function: $w_0 = -0.3$, $w_1 = w_2 = 0.5$

- Perceptrons can represent all of the *primitive* boolean functions AND, OR, NAND, and NOR.
  - Note: Some boolean functions cannot be represented by a *single* perceptron (e.g. XOR). Why not?

- Every boolean function can be represented by some *network* of perceptrons only *two levels* deep. How?
  - One way is to represent the boolean function in DNF form (OR of ANDs).

Perceptron Training Rule

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

where

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

Where:

- $t = c(\vec{x})$ is target value
- o is perceptron output
- η is small constant (e.g., .1) called *learning rate*

- Note: output value *o* is +1 or -1 (not a real)
- Perceptron rule: a learning rule for a *threshold* unit.
- Conditions for convergence
  - Training examples are linearly separable.
  - Learning rate is sufficiently small.

Least Mean Square (LMS) Error

To understand, consider simpler *linear unit*, where

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

Let's learn w_i's that minimize the squared error

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

Where D is set of training examples

- Note: output value *o* is a real value (not binary)
- Delta rule: learning rule for an unthresholded perceptron (*i.e.* linear unit).
  - Delta rule is a gradient-descent rule.
  - Also known as the *Widrow-Hoff* rule

Gradient Descent Method
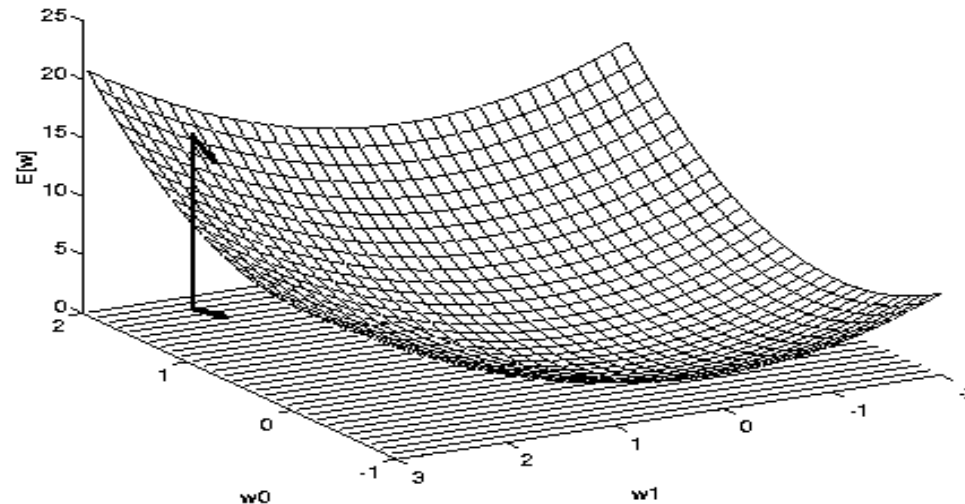
Gradient

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

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

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

Delta Rule for Error Minimization

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d})$$

$$\frac{\partial E}{\partial w_i} = \sum_d (t_d - o_d)(-x_{i,d})$$

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

Gradient Descent Algorithm for Perceptron Learning

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. η 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 Δw_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
      $$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$$
  - For each linear unit weight w_i, Do
    $$w_i \leftarrow w_i + \Delta w_i$$

Properties of Gradient Descent

- Because the error surface contains only a single global minimum, the **gradient descent algorithm** will converge to a weight vector with minimum error, regardless of whether the training examples are linearly separable.
  - Condition: a sufficiently small learning rate
- **If the learning rate is too large**, the gradient descent search may overstep the minimum in the error surface.
  - A solution: gradually reduce the learning rate value.

Conditions for Gradient Descent

- **Gradient descent** is an important general strategy for searching through a large or infinite hypothesis space.

- Conditions for gradient descent search
  - The hypothesis space contains continuously parameterized hypotheses (*e.g.,* the weights in a linear unit).
  - The error can be differentiated w.r.t. these hypothesis parameters.

Difficulties with Gradient Descent

- Converging to a local minimum can sometimes be quite slow (many thousands of gradient descent steps).

- If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

Perceptron Rule *v.s.* Delta Rule

- Perceptron rule
  - Thresholded output
  - Converges after a finite number of iterations to a hypothesis that perfectly classifies the training data, provided the training examples are linearly separable.
  - Can deal with only linearly separable data
- Delta rule
  - Unthresholded output
  - Converges only asymptotically toward the error minimum, possibly requiring unbounded time, but converges regardless of whether the training data are linearly separable.
  - Can deal with linearly nonseparable data

Multilayer Perceptron

Multilayer **N**etwork and
Its Decision Boundaries
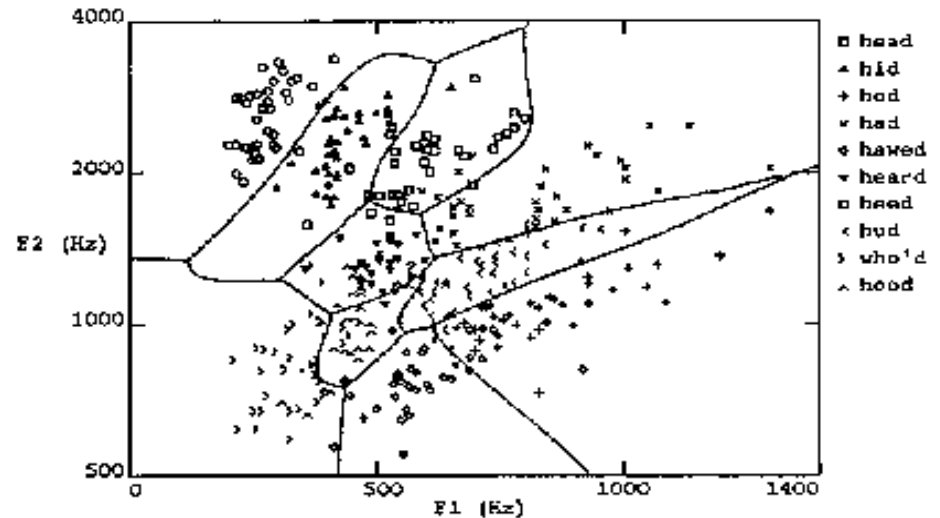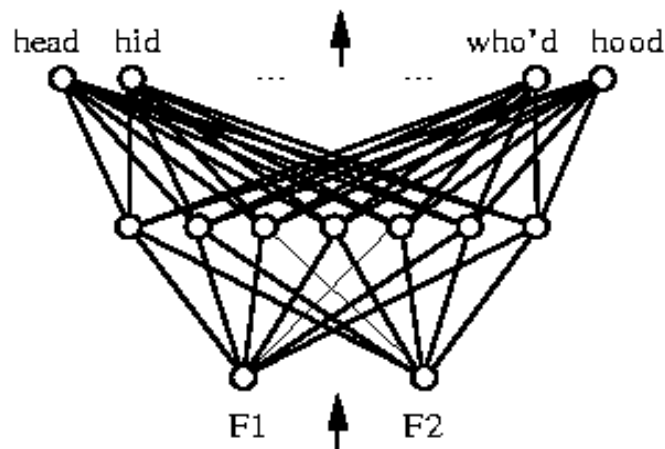
❖ Decision regions of a multilayer feedforward network.

❖ The network was trained to recognize 1 of 10 vowel sounds occurring in the context "h_d"

❖ The network input consists of two parameter, F1 and F2, obtained from a spectral analysis of the sound.

❖ The 10 network outputs correspond to the 10 possible vowel sounds.

Differentiable Threshold Unit

x_1 ⬤ w_1 $x_0 = 1$

x_2 ⬤ w_2 w_0

⋮

w_n

x_n ⬤

Σ → $net = \sum_{i=0}^{n} w_i x_i$ → $o = \sigma(net) = \dfrac{1}{1 + e^{-net}}$
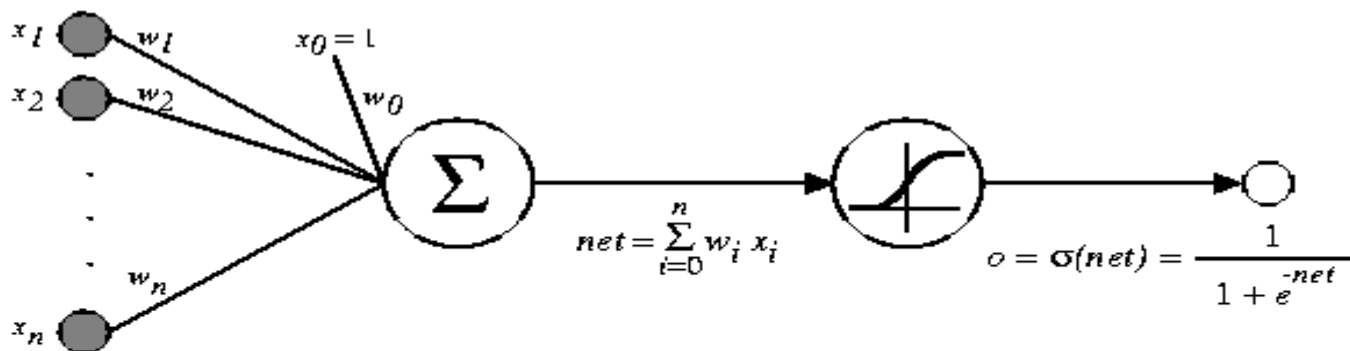
$\sigma(x)$ is the sigmoid function

$$\frac{1}{1 + e^{-x}}$$

Nice property: $\dfrac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x))$

- Sigmoid function: nonlinear, differentiable

Backpropagation (BP) Algorithm

- BP learns the weights for a multilayer network, given a network with a fixed set of units and interconnections.

- BP employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.

- Two stage learning
  - forward stage: calculate outputs given input pattern x.
  - backward stage: update weights by calculating delta.

Error Function for BP

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

- *E* defined as a sum of the squared errors over *all* the output units *k* for *all* the training examples *d*.

- Error surface can have multiple local minima
  - Guarantee toward some local minimum
  - No guarantee to the global minimum

Backpropagation Algorithm for MLP

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do

    1. Input the training example to the network and compute the network outputs

    2. For each output unit k

    $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

    3. For each hidden unit h

    $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

    4. Update each network weight $w_{i,j}$

    $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

    where

    $$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Termination Conditions for BP

- The weight update loop may be iterated thousands of times in a typical application.
- The choice of termination condition is important because
  - Too few iterations can fail to reduce error sufficiently.
  - Too many iterations can lead to overfitting the training data.

- Termination Criteria
  - After a fixed number of iterations (epochs)
  - Once the error falls below some threshold
  - Once the validation error meets some criterion

Adding Momentum

- Original weight update rule for BP:

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji}$$

- Adding momentum α

$$\Delta w_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta w_{ji}(n-1), \qquad 0 < \alpha < 1$$

  - Help to escape a small local minima in the error surface.
  - Speed up the convergence.

Derivation of the BP Rule

- Notations

  - x_{ij} : the ith input to unit j

  - w_{ij} : the weight associated with the ith input to unit j

  - net_j : 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 BP Rule

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

- Gradient descent: $\Delta w_{ji} = -\eta \dfrac{\partial E_d}{\partial w_{ji}}$

- Chain rule: $\dfrac{\partial E_d}{\partial w_{ji}} = \dfrac{\partial E_d}{\partial net_j} \dfrac{\partial net_j}{\partial w_{ji}} = \dfrac{\partial E_d}{\partial net_j} x_{ji}$

Case 1: Rule for Output Unit Weights

- Step 1:

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

- Step 2:

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

- Step 3:

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

- All together:

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

Case 2: Rule for Hidden Unit Weights

- Step 1:
$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j}$$

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

- Thus:
$$\Delta w_{ji} = \eta \delta_j x_{ji}, \quad \text{where} \quad \delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj}$$

Backpropagation for MLP: revisited

Initialize all weights to small random numbers.
Until satisfied, Do

- For each training example, Do

  1. Input the training example to the network and compute the network outputs

  2. For each output unit k

  $$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$$

  3. For each hidden unit h

  $$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{h,k}\delta_k$$

  4. Update each network weight $w_{i,j}$

  $$w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$$

  where

  $$\Delta w_{i,j} = \eta \delta_j x_{i,j}$$

Convergence and Local Minima

- The error surface for multilayer networks may contain many different local minima.
- BP guarantees to converge local minima only.
- BP is a highly effective function approximator in practice.
  - The local minima problem found to be not severe in many applications.

Notes
  - Gradient descent over the complex error surfaces represented by ANNs is still poorly understood
  - No methods are known to predict certainly when local minima will cause difficulties.
  - We can use only heuristics for avoiding local minima.

Heuristics for Alleviating the Local Minima Problem

- Add a momentum term to the weight-update rule.

- Use stochastic descent rather than true gradient descent.
  - Descend a different error surface for each example.

- Train multiple networks using the same data, but initializing each network with different random weights.
  - Select the best network w.r.t the validation set
  - Make a committee of networks

Why BP Works in Practice?

A Possible Senario

- Weights are initialized to values near zero.
- Early gradient descent steps will represent a very smooth function (approximately linear). Why?
  - The sigmoid function is almost linear when the total input (weighted sum of inputs to a sigmoid unit) is near 0.
- The weights gradually move close to the global minimum.
- As weights grow in the later stage of learning, they represent highly nonlinear network functions.
- Gradient steps in this later stage move toward local minima in this region, which is acceptable.

Representational Power of MLP

- Every boolean function can be represented exactly by some network with two layers of units. How?

  – Note: The number of hidden units required may grow exponentially with the number of network inputs.

- Every bounded continuous function can be approximated with arbitrarily small error by a network of two layers of units.

  – Sigmoid hidden units, linear output units

  – How many hidden units?

NNs as Universal Function Approximators

- Any function can be approximated to arbitrary accuracy by a network with *three* layers of units (Cybenko 1988).
  - Sigmoid units at two hidden layers
  - Linear units at the output layer
  - Any function can be approximated by a linear combination of many localized functions having 0 everywhere except for some small region.
  - Two layers of sigmoid units are sufficient to produce good approximations.

- Every bounded continuous function can be approximated, with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]

BP Compared with CE & ID3

- For BP, every possible assignment of network weights represents a syntactically distinct hypothesis.
  - The hypothesis space is the n-dimensional Euclidean space of the n network weights.

- Hypothesis space is continuous
  - The hypothesis space of CE and ID3 is discrete.

- Differentiable
  - Provides a useful structure for gradient search.
  - This structure is quite different from the general-to-specific ordering in CE, or the simple-to-complex ordering in ID3 or C4.5.

CE: candidate-elimination algorithm in 'concept learning' (T.M. Mitchell)
ID3: a learning scheme of 'Decision Tree' for discrete values (R. Quinlan)
C4.5: an improved scheme of ID3 for 'real values' (R. Quinlan)

Hidden Layer Representations

- BP has an ability to discover useful intermediate representations at the hidden unit layers inside the networks which capture properties of the input spaces that are most relevant to learning the target function.

- When more layers of units are used in the network, more complex features can be invented.

- But the representations of the hidden layers are very hard to understand for human.

Hidden Layer Representation for Identity Function

A network:
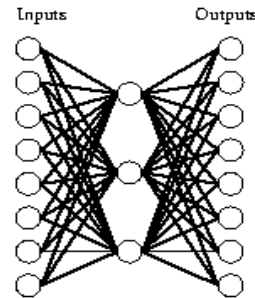
Learned hidden layer representation:

Input	Hidden Values			Output
10000000 →	.89	.04	.08 →	10000000
01000000 →	.01	.11	.88 →	01000000
00100000 →	.01	.97	.27 →	00100000
00010000 →	.99	.97	.71 →	00010000
00001000 →	.03	.05	.02 →	00001000
00000100 →	.22	.99	.99 →	00000100
00000010 →	.80	.01	.98 →	00000010
00000001 →	.60	.94	.01 →	00000001

Hidden Layer Representation for Identity Function

Sum of squared errors for each output unit
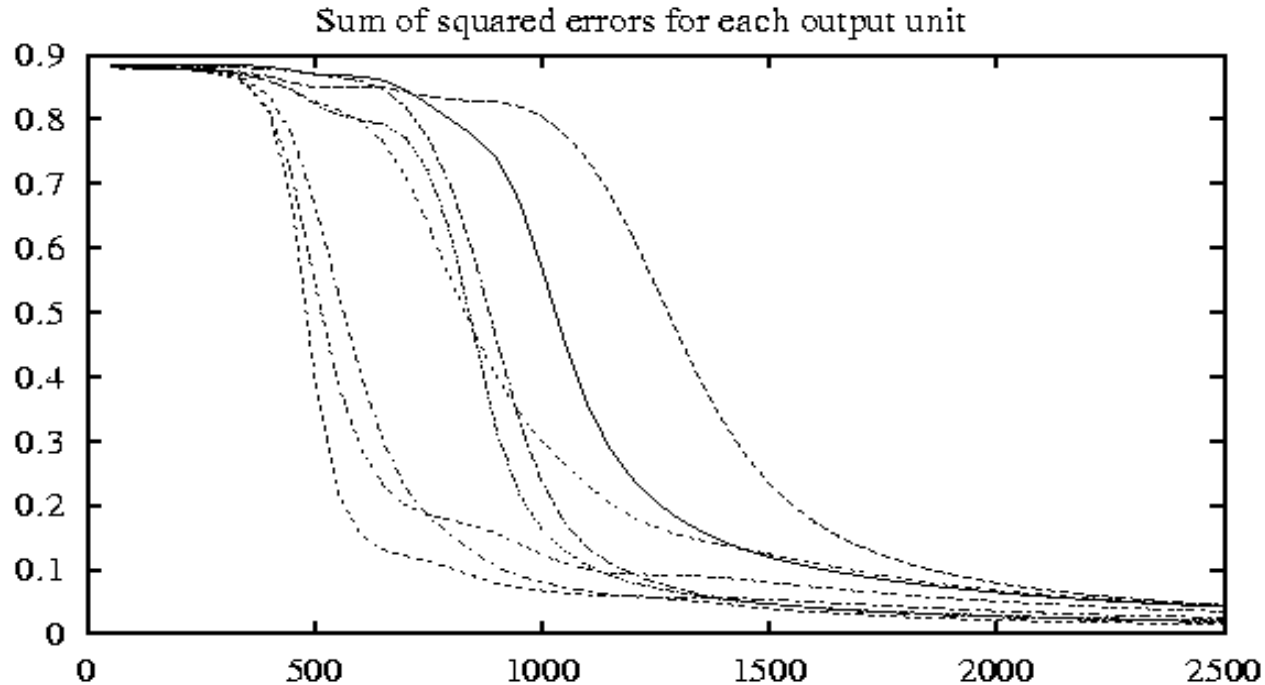
❖ The evolving sum of squared errors for each of the eight output units as the number of training iterations (epochs) increase

Hidden Layer Representation for Identity Function

Hidden unit encoding for input 01000000
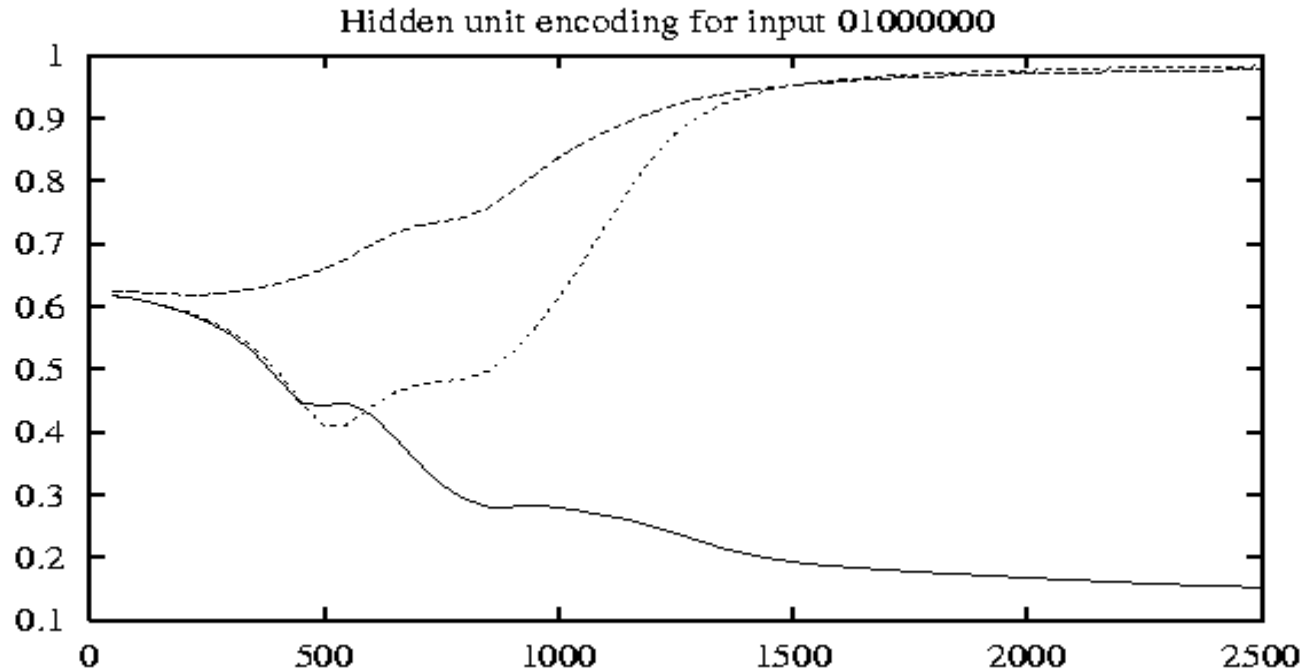
❖ The evolving hidden layer representation for the input string "01000000"

Hidden Layer Representation for Identity Function

Weights from inputs to one hidden unit
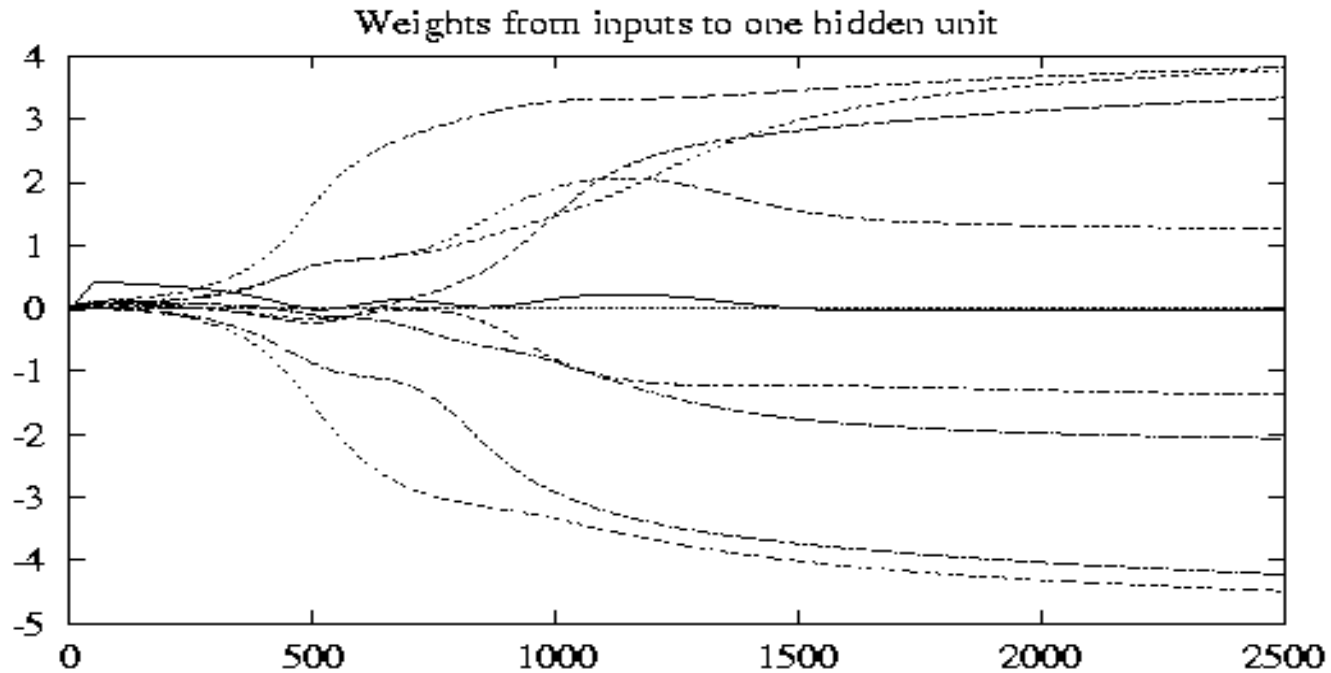
❖ The evolving weights for one of the three hidden units

Generalization and Overfitting

- Continuing training until the training error falls below some predetermined threshold is a poor strategy since BP is susceptible to overfitting.
  - Need to measure the generalization accuracy over a validation set (distinct from the training set).
- Two different types of overffiting
  - Generalization error first decreases, then increases, even the training error continues to decrease.
  - Generalization error decreases, then increases, then decreases again, while the training error continues to decreases.

Two Kinds of Overfitting Phenomena

Error versus weight updates (example 1)

Error versus weight updates (example 2)
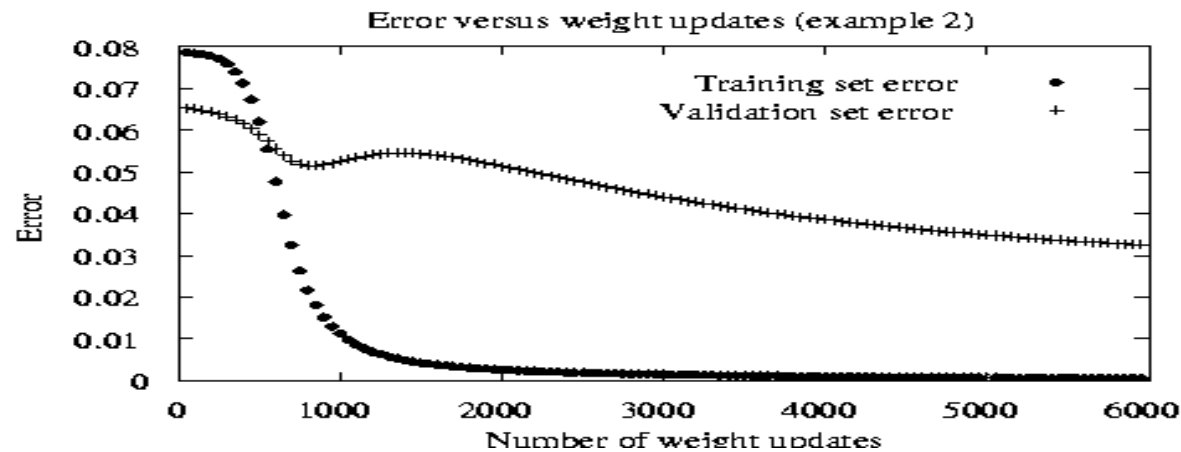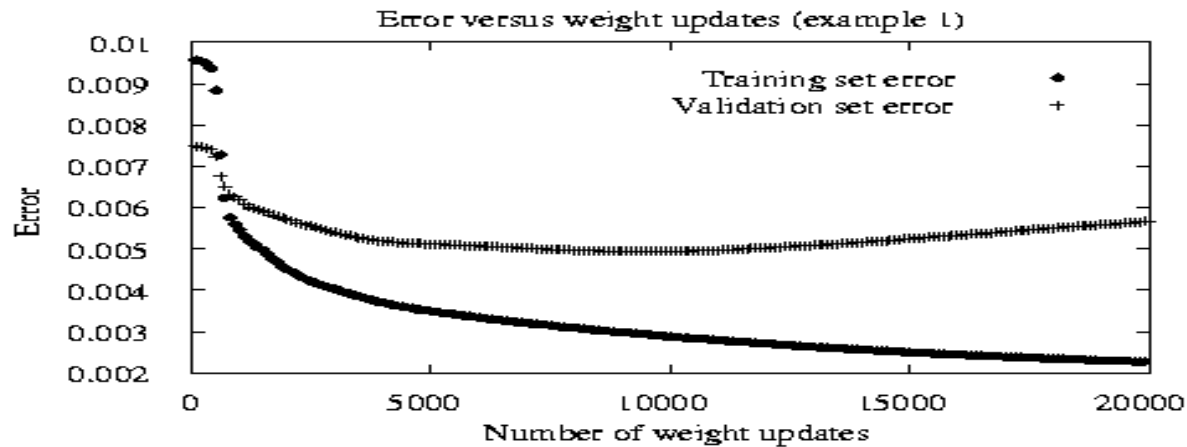
Techniques for Overcoming the Overfitting Problem

- Weight decay
  - Decrease each weight by some small factor during each iteration.
  - This is equivalent to modifying the definition of E to include a penalty term corresponding to the total magnitude of the network weights.
  - The motivation for the approach is to keep weight values small, to bias learning against complex decision surfaces.
- *k*-fold cross-validation
  - Cross validation is performed *k* different times, each time using a different partitioning of the data into training and validation sets
  - The result are averaged after *k* times cross validation.

Designing an Artificial Neural Network for Face Recognition Application

Problem Definition

- Possible learning tasks
  - Classifying camera images of faces of people in various poses.
  - Direction, Identity, Gender, …
- Data:
  - 624 grayscale images for 20 different people
  - 32 images per person, varying
    - person's expression (happy, sad, angry, neutral)
    - direction (left, right, straight ahead, up)
    - with and without sunglasses
  - resolution of images: 120 x128, each pixel with a grayscale intensity between 0 (black) and 255 (white)
- Task: Learning the direction in which the person is facing.

Factors for ANN Design in the Face Recognition Task

Input encoding

Output encoding

Network graph structure

Other learning algorithm parameters

Input Coding for Face Recognition

- Possible Solutions
  - Extract key features using preprocessing
  - Coarse-resolution
- Features extraction
  - edges, regions of uniform intensity, other local image features
  - Defect: High preprocessing cost, variable number of features
- Coarse-resolution
  - Encode the image as a fixed set of 30 x 32 pixel intensity values, with one network input per pixel.
  - The 30x32 pixel image is a coarse resolution summary of the original 120x128 pixel image
  - Coarse-resolution reduces the number of inputs and weights to a much more manageable size, thereby reducing computational demands.

Output Coding for Face Recognition

- Possible coding schemes
  - Using one output unit with multiple threshold values
  - Using multiple output units with single threshold value.
- One unit scheme
  - Assign 0.2, 0.4, 0.6, 0.8 to encode four-way classification.
- Multiple units scheme (*1-of-n* output encoding)
  - Use four distinct output units
  - Each unit represents one of the four possible face directions, with highest-valued output taken as the network prediction

Output Coding for Face Recognition

- Advantages of *1-of-n* output encoding scheme
  - It provides more degrees of freedom to the network for representing the target function.
  - The difference between the highest-valued output and the second-highest can be used as a measure of the confidence in the network prediction.
- Target value for the output units in *1-of-n* encoding scheme
  - < 1, 0, 0, 0 > *v.s.* < 0.9, 0.1, 0.1, 0.1 >
  - < 1, 0, 0, 0 >: will force the weights to grow without bound.
 - < 0.9, 0.1, 0.1, 0.1 >: the network will have finite weights.

Network Structure for Face Recognition

- One hidden layer *v.s.* more hidden layers

- How many hidden nodes is used?
  - Using 3 hidden units:
    - test accuracy for the face data = 90%
    - Training time = 5 min on Sun Sprac 5
  - Using 30 hidden units:
    - test accuracy for the face data = 91.5%
    - Training time = 1 hour on Sun Sparc 5

Other Parameters for Face Recognition

- Learning rate $\eta = 0.3$
- Momentum $\alpha = 0.3$
- Weight initialization: small random values near 0
- Number of iterations: Cross validation
  - After every 50 iterations, the performance of the network was evaluated over the validation set.
  - The final selected network is the one with the highest accuracy over the validation set

ANN for Face Recognition
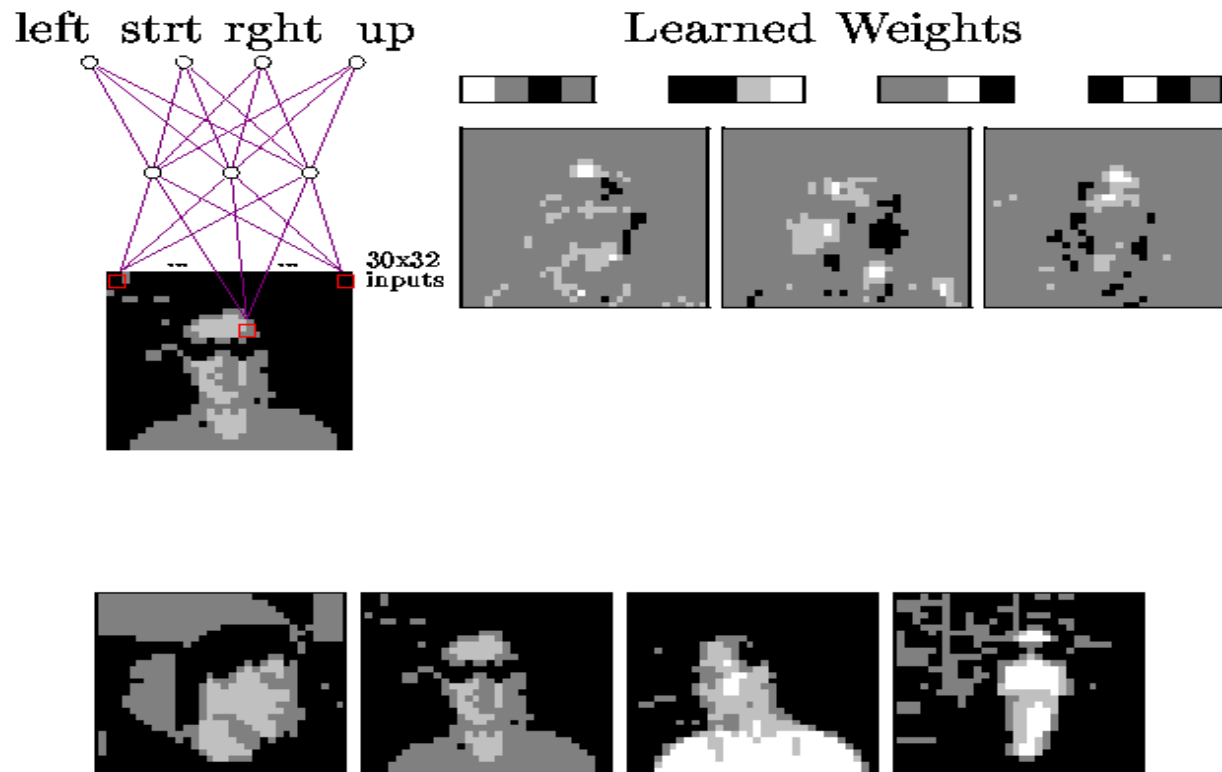
960 x 3 x 4 network is trained on gray-level images of faces to predict whether a person is looking to their left, right, ahead, or up.

Questions !

- Do I need to code all the math ?
- Won't it be better if we took advantage of parallel processing capabilities we have now ?
- Are there any tools available to assist me in implementing ANN for my problem ?
- Do I need a super computer to do call the Calculations required ?

Answer