Artificial Neural Networks

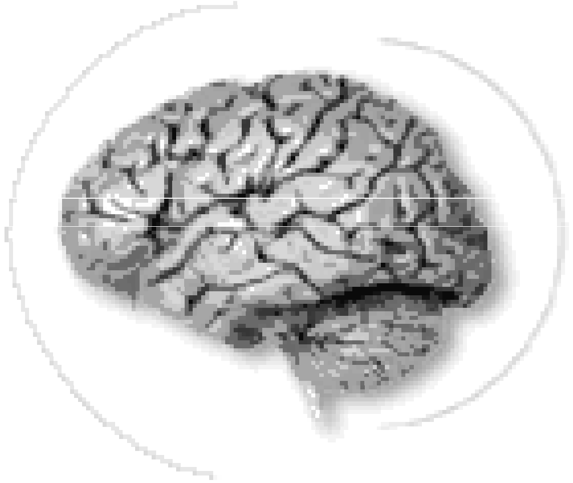
장 병 탁
서울대 컴퓨터공학부
E-mail: btzhang@cse.snu.ac.kr
http://bi.snu.ac.kr./~btzhang/

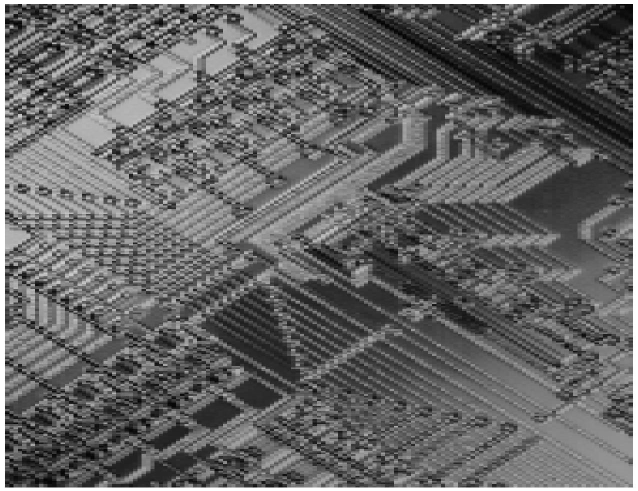
Byoung-Tak Zhang
School of Computer Science and Engineering
Seoul National University

Outline

1. Basic Concepts of Neural Networks

The Brain vs. Computer





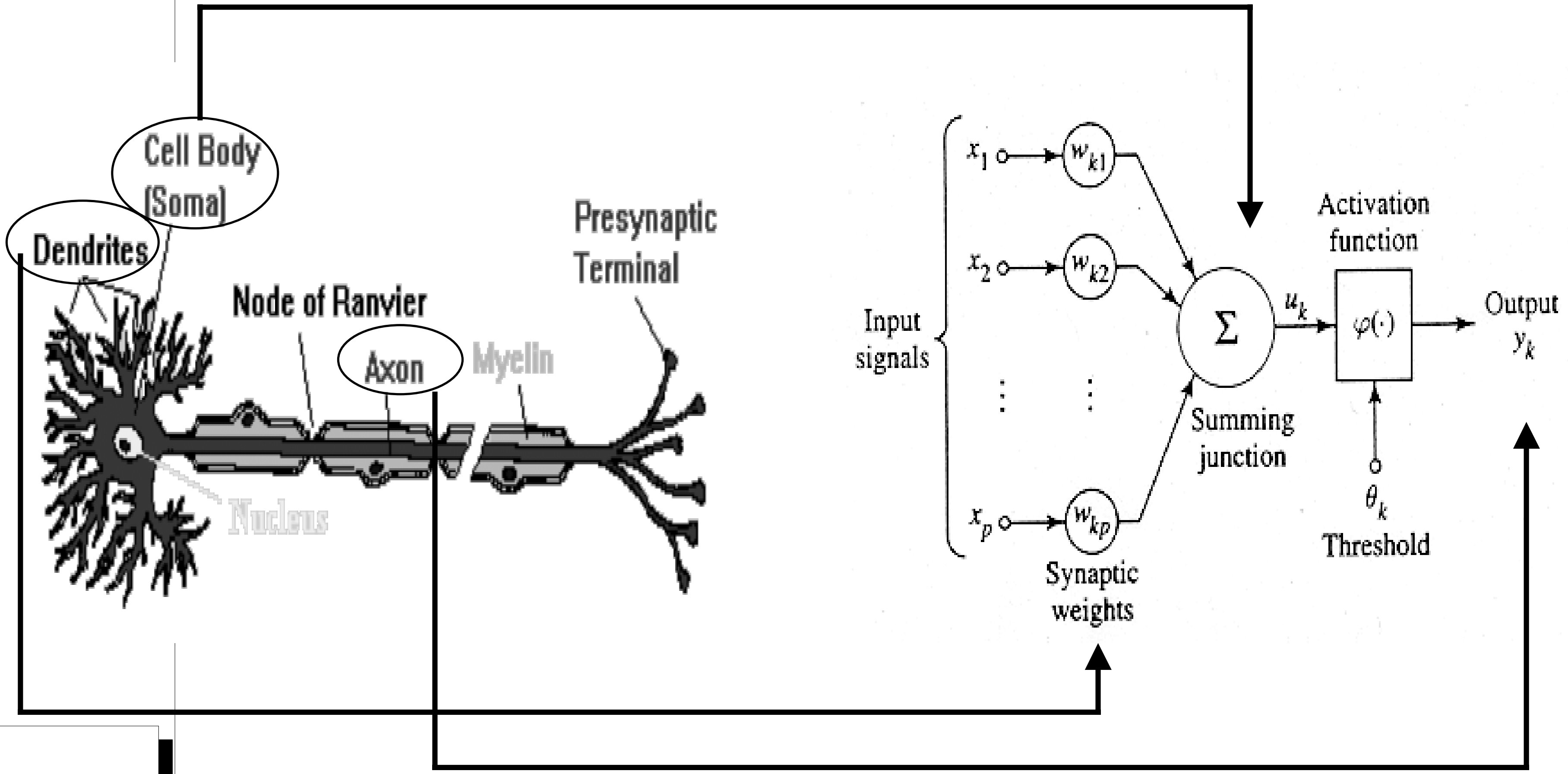
1. 10^{11} neurons with 10^{14} synapses
2. Speed: 10^{-3} sec
3. Distributed processing
4. Nonlinear processing
5. Parallel processing

1. A single processor with complex circuits
2. Speed: 10^{-9} sec
3. Central processing
4. Arithmetic operation (linearity)
5. Sequential processing

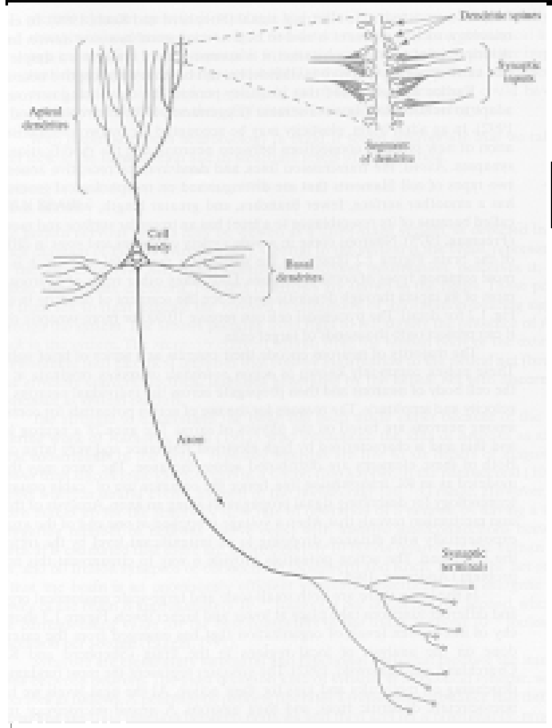
What Is a Neural Network?

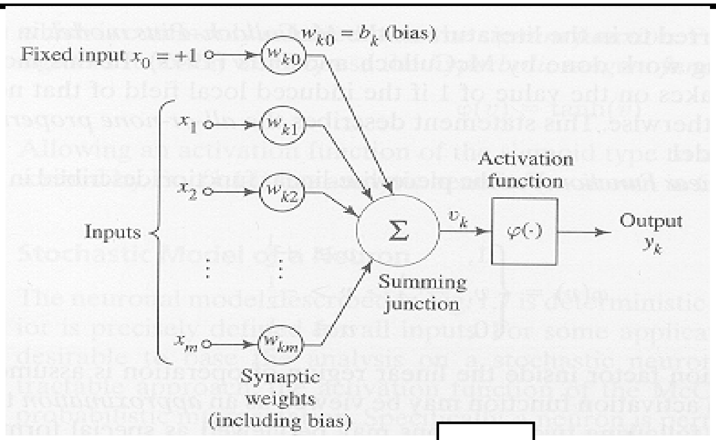
- A new form of computing, inspired by biological (brain) models.
- A mathematical model composed of a large number of simple, highly interconnected processing elements.
- A computational model for studying learning and intelligence.

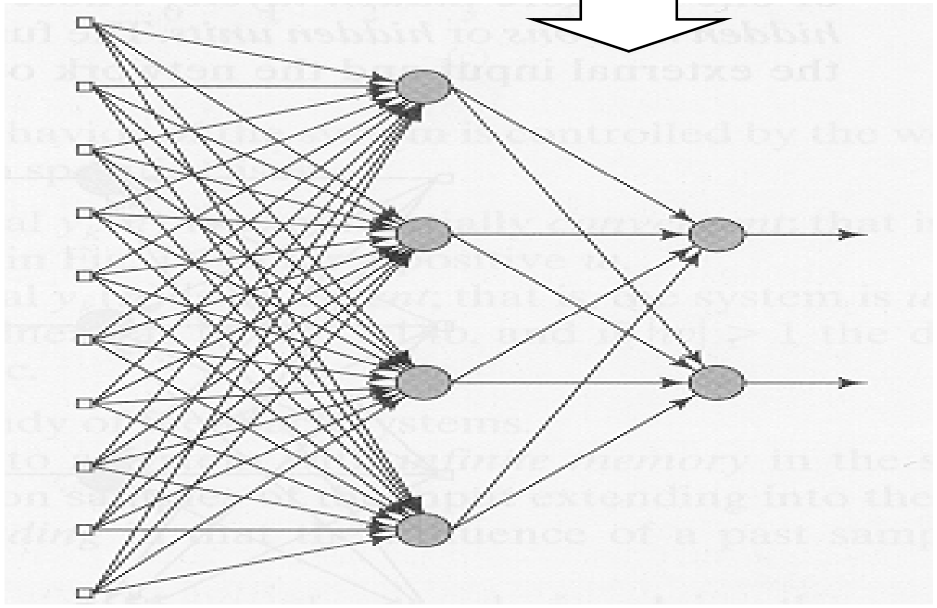
From Biological Neuron to Artificial Neuron



From Biology to Artificial Neural Networks (ANNs)

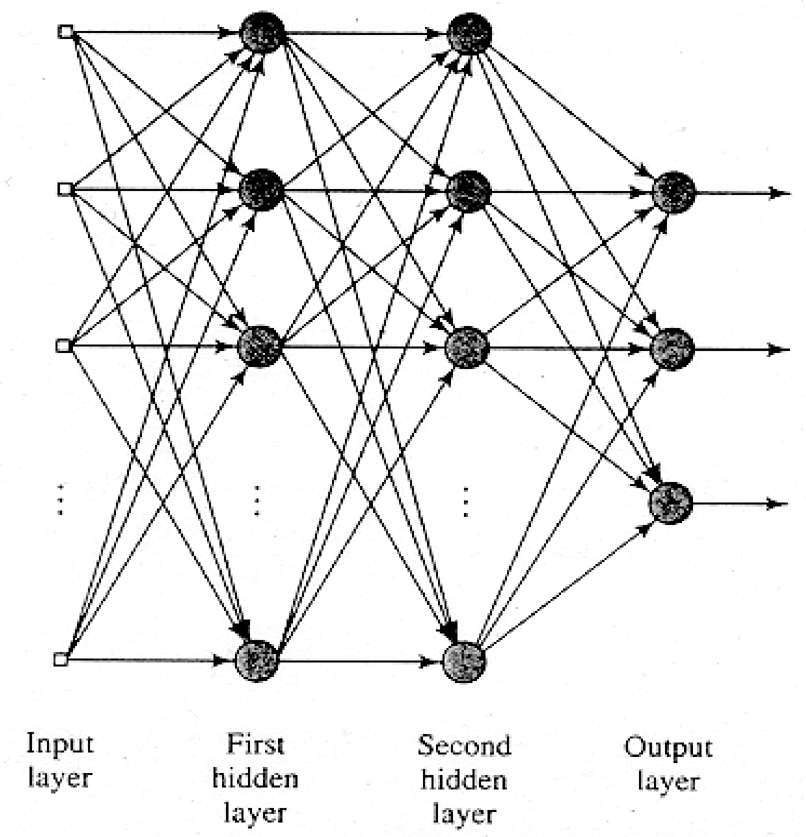






Properties of Artificial Neural Networks

- A network of artificial neurons



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

Synonyms for Neural Networks

- Neurocomputing
- Neuroinformatics (Neuroinformatik)
- Neural Information Processing Systems
- Connectionist Models
- Parallel Distributed Processing (PDP) Models
- Self-organizing Systems
- Neuromorphic Systems

Brief History

- William James (1890): Describes (in words and figures) simple distributed networks and Hebbian Learning
- McCulloch & Pitts (1943): Binary *threshold* units that perform logical operations (they prove universal computations!)
- Hebb (1949): Formulation of a physiological (local) learning rule
- Rosenblatt (1958): The Perceptron - a first real learning machine
- Widrow & Hoff (1960): ADALINE and the Windrow-Hoff supervised learning rule.
- Minsky & Papert (1969): The limitations of perceptron – the beginning of the “Neural Winter”
- v.d.Malsburg (1973): Self-organizing Maps
- Grossberg (1980): Adaptive Resonance Theory
- Hopfield (1982/84): Attractor Networks: A clean theory of pattern association and memory
- Kohonen (1982): Self-organizing maps.
- Rumelhart, Hinton, & Williams (1986): Backprop

Types of NNs

- Single Layer Perceptrons
- Multilayer Perceptrons (MLPs)
- Radial-Basis Function Networks (RBFs)
- Hopfield Networks
- Boltzmann Machines
- Self-Organization Maps (SOMs)
- Modular Networks (Committee Machines)
- Support Vector Machines
- Bayesian Networks
- Probabilistic Graphical Models
- Hidden Markov Models

Neural Network Models

- Unit types
  - ♦ Deterministic / stochastic units
  - ♦ Linear / threshold / sigmoid units
  - ♦ Gaussian / softmax units
- Network topology
  - ♦ Feedforward / recurrent networks
  - ♦ Layered / non-layered networks
  - ♦ Fully / partially connected networks
- Learning methods
  - ♦ Supervised / unsupervised / reinforcement learning
  - ♦ Batch / on-line training

Unit Types: Activation Functions

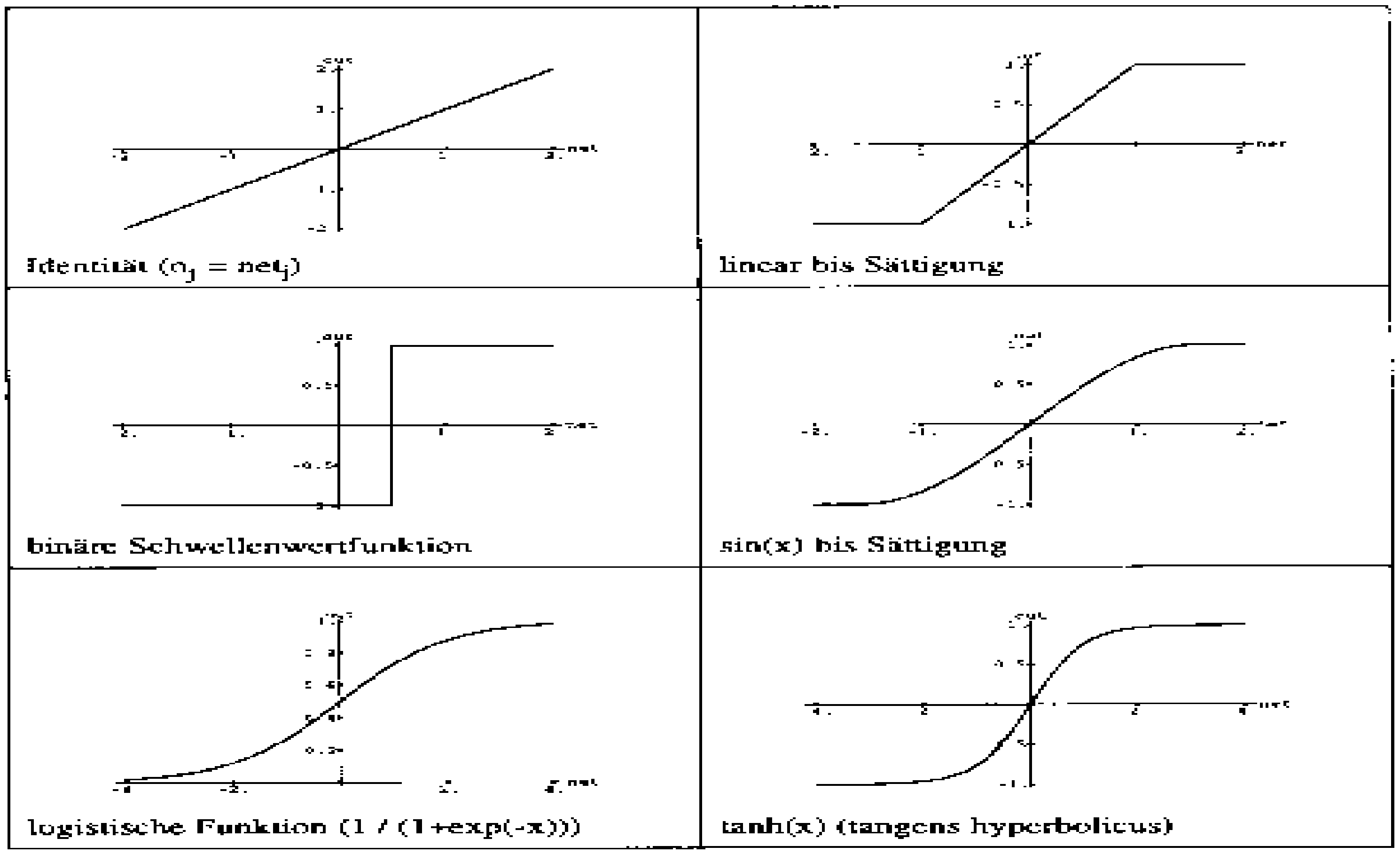
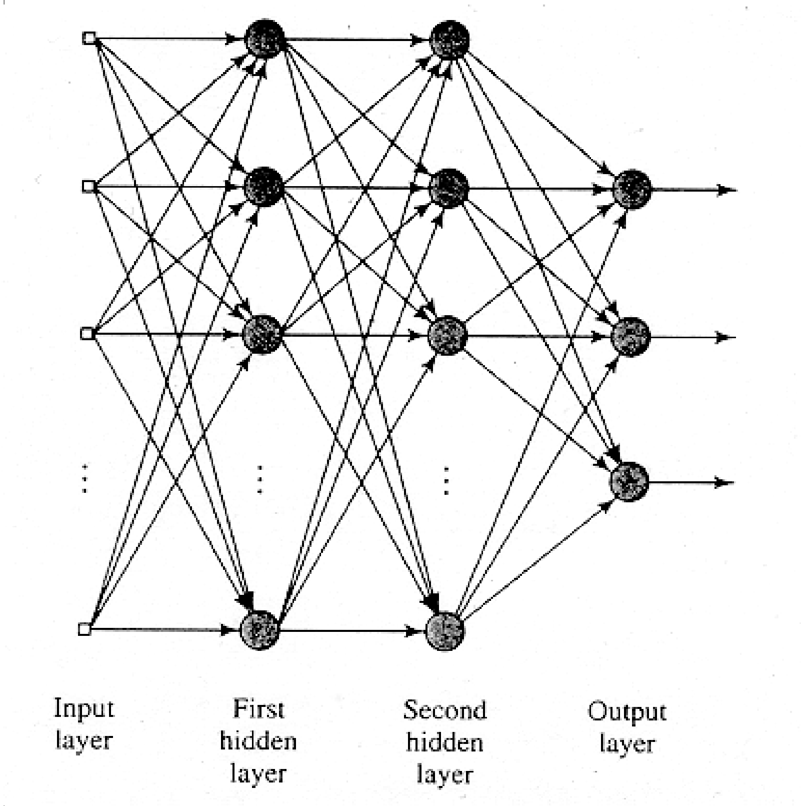


Abb. 5.6: Häufig verwendete Aktivierungs- bzw. Ausgabefunktionen.

Network Topology



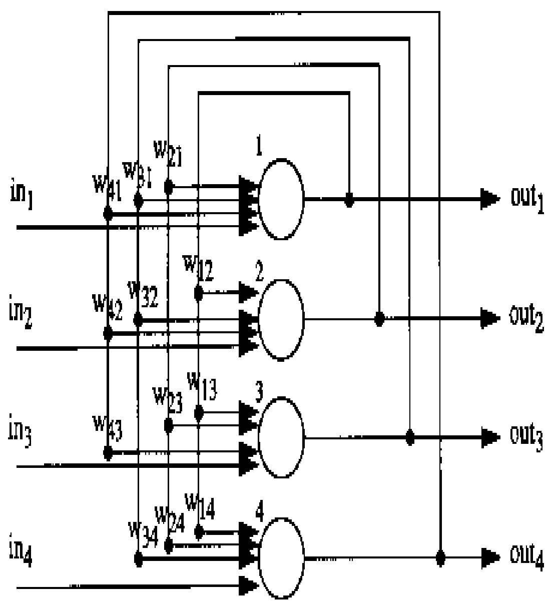
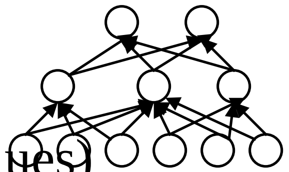


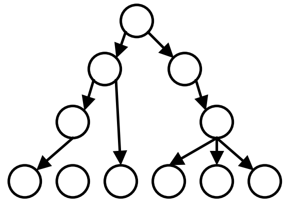
Abb. 17.2: Hopfield-Netz mit 4 Neuronen

Supervised vs. Unsupervised Learning

- Supervised Learning
  - Estimate an unknown mapping from known input- output pairs
  - Learn $f_{\mathbf{w}}$ from training set $D=\{(\mathbf{x},y)\}$ s.t. $f_{\mathbf{w}}(\mathbf{x}) = y = f(\mathbf{x})$
  - Classification: y is discrete
  - Regression: y is continuous
  - Example: Hand-written numeral recognition
    - $\mathbf{x}$: a scanned numeral (vector of gray-scale values)
    - y: class of the numeral (0, 1, …, or 9)



- Unsupervised Learning
  - Only input values are provided
  - Learn $f_{\mathbf{w}}$ from $D=\{(\mathbf{x})\}$ s.t. $f_{\mathbf{w}}(\mathbf{x}) = \mathbf{x}$
  - Compression, clustering, density estimation



- Reinforcement Learning

Issues in Neural Networks

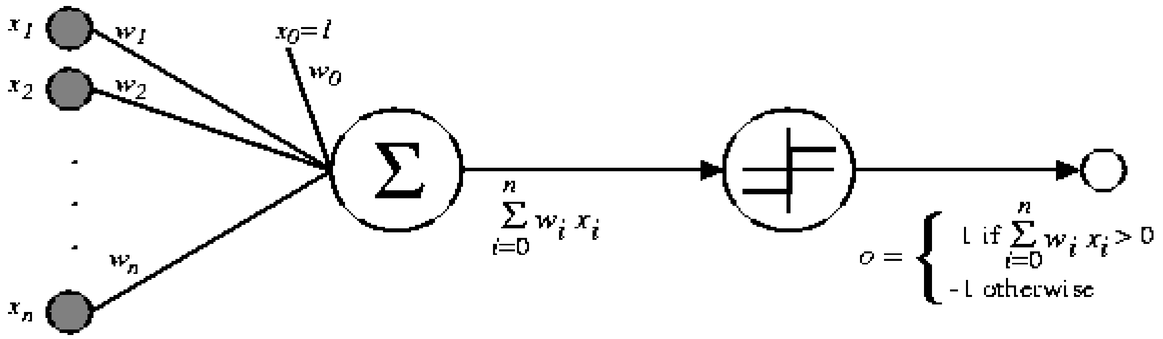
- What is an appropriate architecture for a given learning problem?
  - Should units be divided into layers? How many?
  - What sort of activation function in units?
  - What type of updating? Incremental (stochastic), batch, synchronous, asynchronous?
  - How many units?
- How can the network be programmed?
  - What must be pre-designed? What can be learned?
  - How many samples are needed for good performance?
  - Is on-line learning possible?
  - What kind of learning information is available?
- What are the characteristics of a network?
  - For which problem is it suitable (what “function” can the network represent)?
  - Is it optimal in some sense?
  - What are the assumptions underlying the network? (structural biases, problem biases, noise distributions, etc.)
  - How robust is a network to violations of its assumptions?
  - Can the network generalize from a known task to unknown ones?
  - How hard is it to initialize the network?

Problems Appropriate for NNs

- A large set of <input, output> pairs are available as training examples.
- Output values are discrete, continuous, or combinations of both.
- Learning examples are noisy.
- Long learning time is tolerable.
- Fast execution is required.
- Human interpretation of the learned model is not important.

2. Simple Perceptron and Delta Rule

Architecture of Simple Perceptron

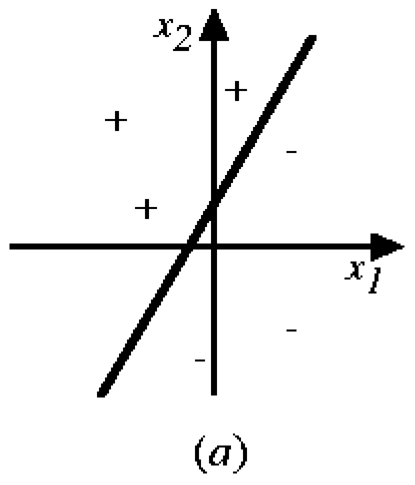


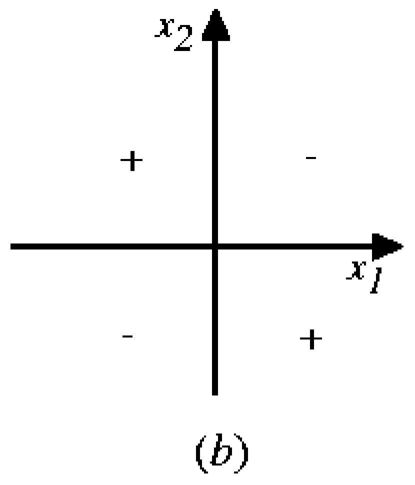
x_1
x_2
x_n
w_1
w_2
w_n
$x_0=1$
w_0
Σ
$\sum_{i=0}^{n} w_i x_i$
$o = \begin{cases} 1 \text{ if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 \text{ otherwise} \end{cases}$

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

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

Linearly Separable *vs.* Linearly Nonseparable





(a) Decision surface for a *linearly separable* set of examples (correctly classified by a straight line). Perceptrons can learn this.

(b) A set of training examples that is *not linearly separable*. Perceptrons are not able to represent this boundary function.

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

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

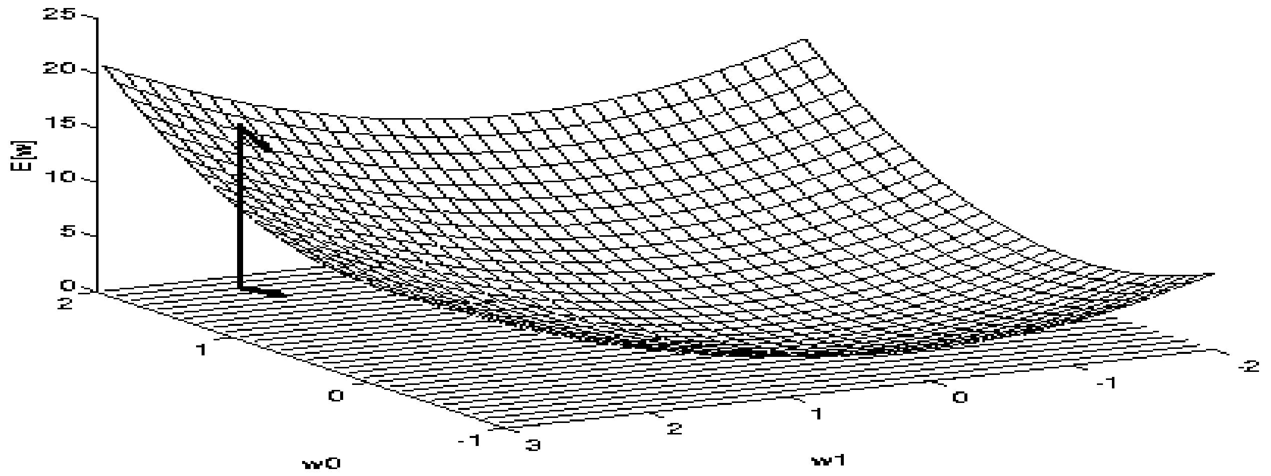
Let's learn w_i's that minimize the squared error

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

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.

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

$$w_i \leftarrow w_i + \Delta w_i \quad , \quad \Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_d (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\
&= \frac{1}{2} \sum_d 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\
&= \sum_d (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x_d}) \\
\frac{\partial E}{\partial w_i} &= \sum_d (t_d - o_d)(-x_{i,d})
\end{aligned}$$

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

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. $\eta$ 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
              $\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i$
    – 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

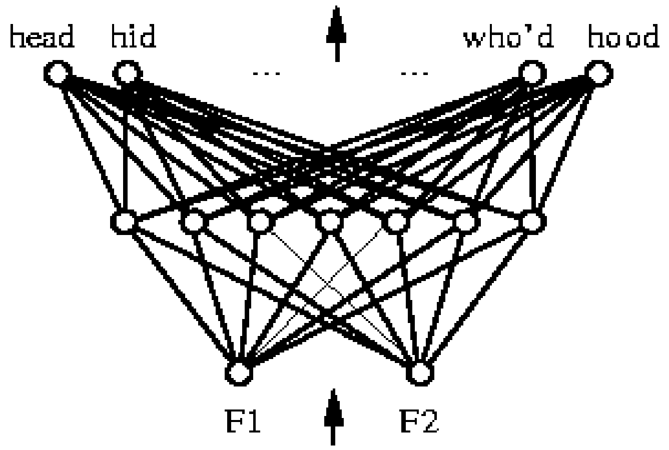
- **If the learning rate is too large**, the gradient descent search may overstep the minimum in the error surface.
  - ♦ A solution: gradually reduce the learning rate value.

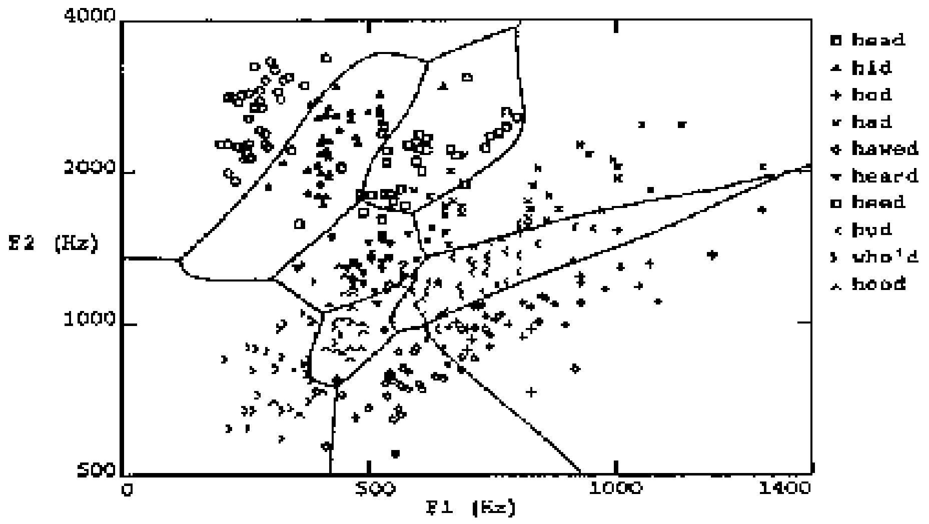
Perceptron Rule *vs.* Delta Rule

- 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.
  - ♦ 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.
  - ♦ Linearly nonseparable data

3. Multilayer Perceptron and Backpropagation Learning

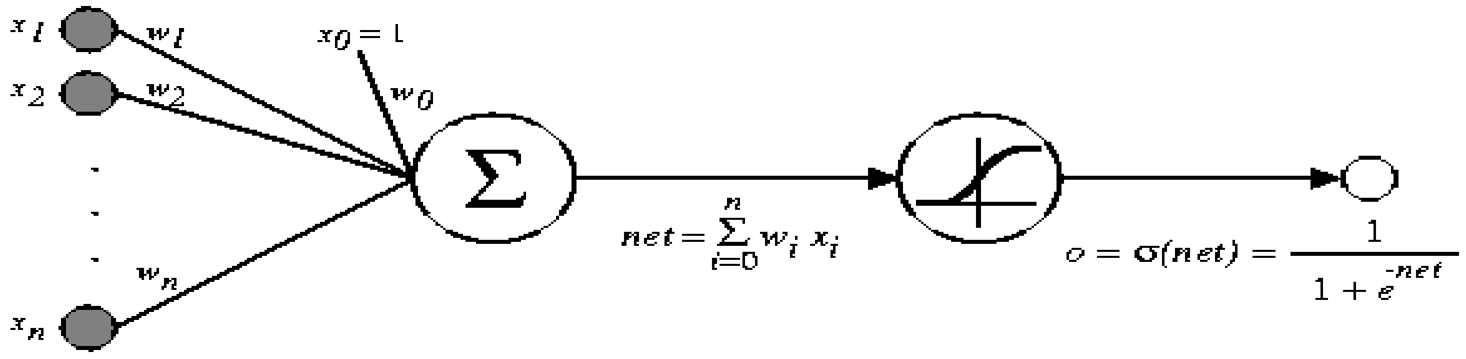
Multilayer Networks and Their Decision Boundaries





- ☒ 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: Sigmoid


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

$\sigma(x)$ **is the sigmoid function**

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

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

- 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 *i*th input to unit *j*
  - ♦ w_{ij} : the weight associated with the *i*th 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: $E_d(\vec{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$

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

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

Case 1: Rule for Output Unit Weights

- Step 1: $\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j}$ $\qquad net_j = \sum_i w_{ji} x_{ji}$

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

BP 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 a 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

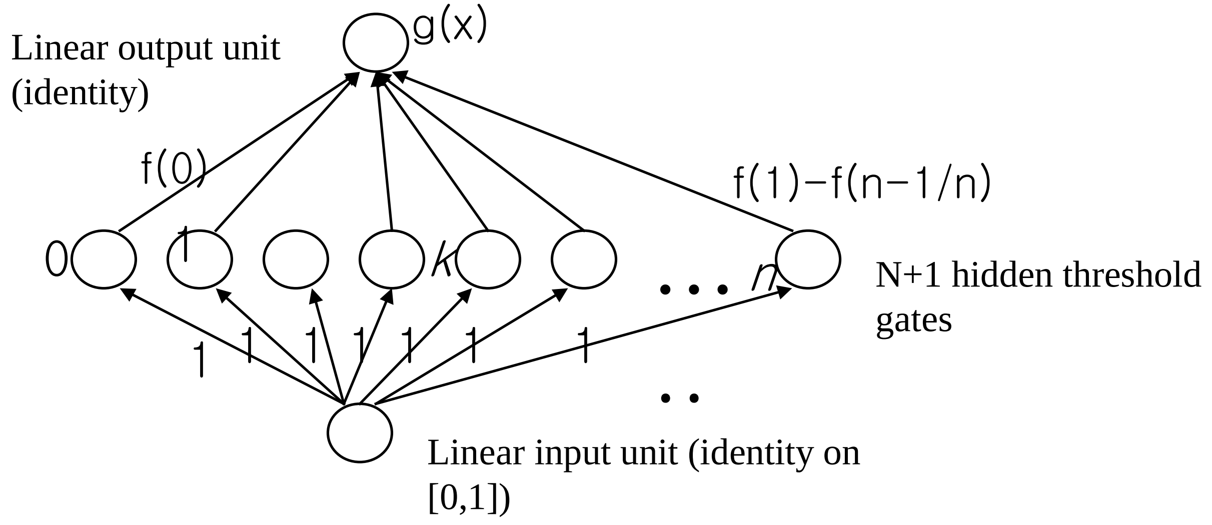
- Every boolean function can be represented exactly by some network with two layers of units. How?
  - Note: The number of hidden units required may grow exponentially with the number of network inputs.
- Every bounded continuous function can be approximated with arbitrarily small error by a network 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.

Universal Approximation Properties

- A three-layer network can approximate any reasonable function to any degree of required precision as long as the hidden layer can be arbitrarily large.
  - Ex) any Boolean function can be built using a combination of threshold gates.



g(x)
Linear output unit (identity)
f(0)
f(1)−f(n−1/n)
0
1
k
n
N+1 hidden threshold gates
1
1
1
1
1
1
1
Linear input unit (identity on [0,1])

Universal approximation architecture with one input unit, *n+1* hidden threshold gate units, and one linear output unit

BP Compared with ID3

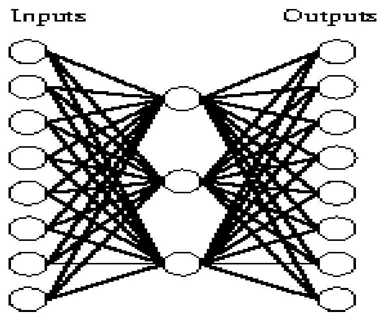
- For BP, every possible assignment of network weights represents a syntactically distinct hypothesis.
  - ♦ The hypothesis space is the *n*-dimensional Euclidean space of the *n* network weights.
- Hypothesis space is continuous
  - ♦ The hypothesis space of candidate elimination (CE) and ID3 is discrete.
- 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.

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

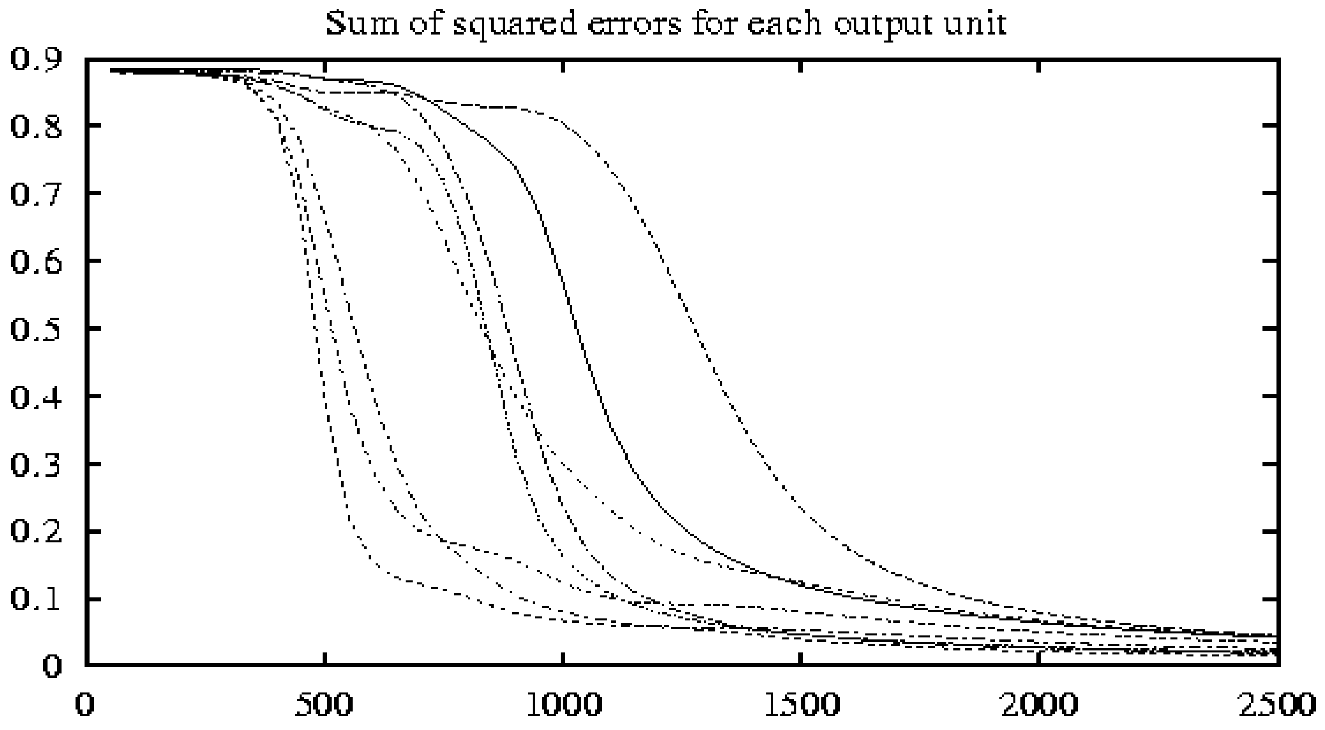
A network:



Learned hidden layer representation:

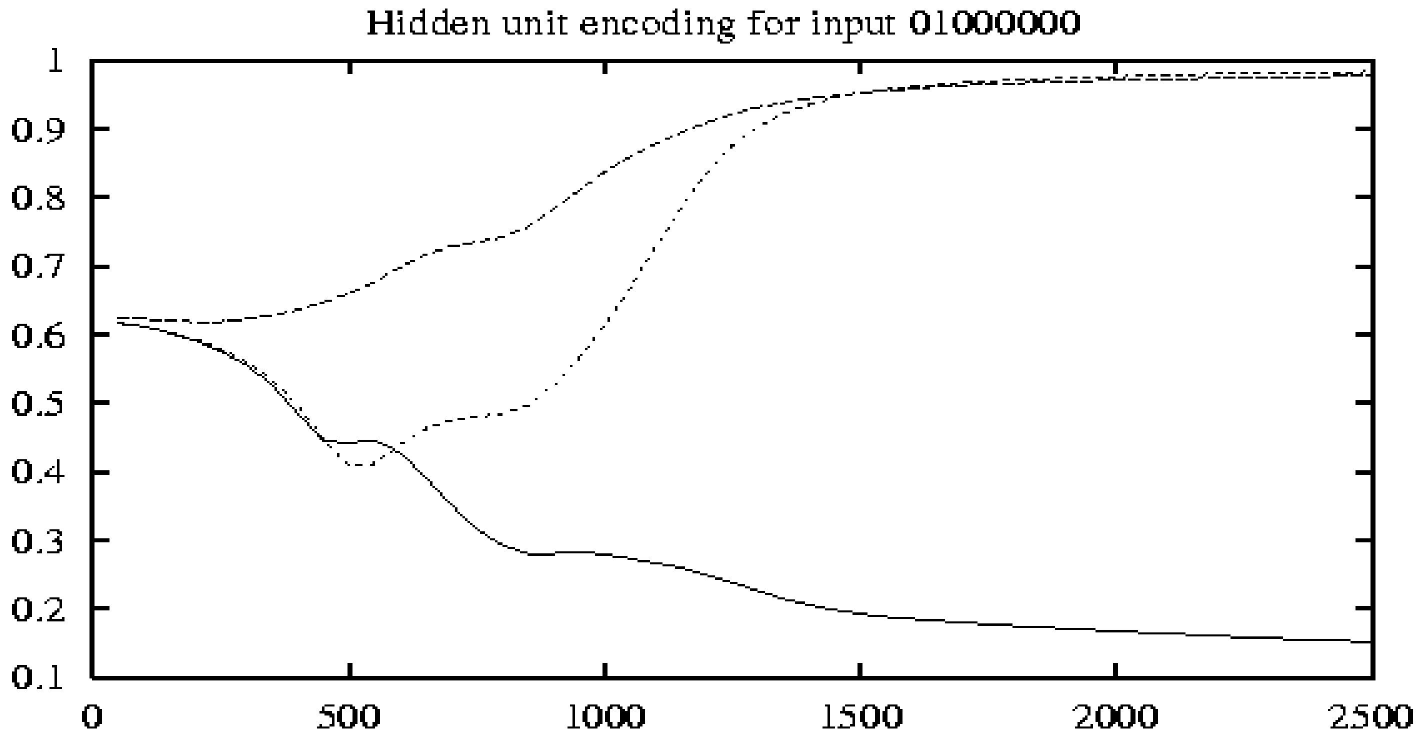
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Hidden Layer Representation for Identity Function



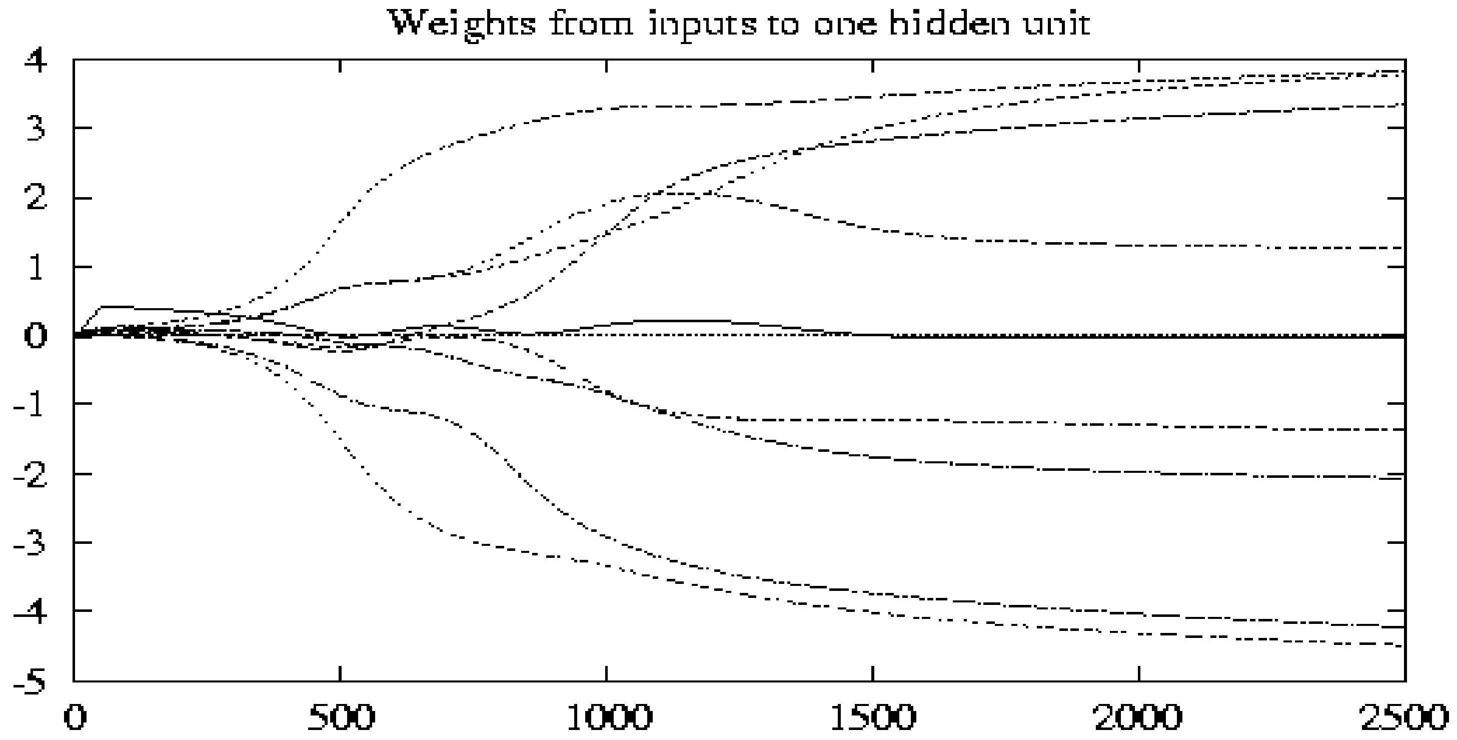
- The evolving sum of squared errors for each of the eight output units as the number of training iterations (epochs) increase

Hidden Layer Representation for Identity Function



- The evolving hidden layer representation for the input string “01000000”

Hidden Layer Representation for Identity Function



- The evolving weights for one of the three hidden units

4. Applications of Neural Networks

Application Examples

- Phoneme recognition
  - Inputs: individual phoneme
  - Outputs: phoneme recognition & classification
- Robot control
  - Inputs: visual & auditive sensor
  - Outputs: robot arm commands
- Weather Forecast
  - Input: precipitation, temperature, pressure
  - Output: forecast, precipitation, temperature, pressure
- Mortgage loan evaluations
  - Input: loan application information
  - Output: loan approval or disapproval
- Financial analysis
  - Input: financial trends & data
  - Output: financial forecasts or assessments
- Image classification
  - Input: images
  - Output: image classes

Application Examples (Cont'd)

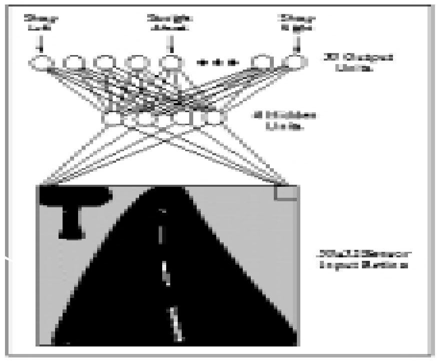
- Automated medical tests
  - Input: reaction development (visual image)
  - Output: medical tests result: presence or absence of reaction
- Automated target recognition
  - Input: image of target
  - Output: target identification
- Pattern recognition
  - Input: partially observed image of target
  - Output: completed picture of target
- Industrial inspection
  - Input: image of industrial parts for inspection
  - Output: fault/ no fault classification
- Dermatology diagnosis
  - Input: dermatology symptoms
  - Output: diagnosis of skin disease
- Jet engine fault diagnosis
  - Input: sensor data engine
  - Output: identification of fault type

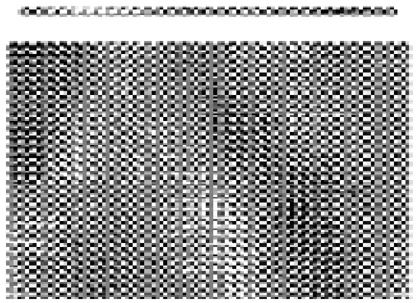
Application Examples (Cont’d)

- Automotive control system diagnosis
  - ♦ Input: automotive engine sensor data and observations
  - ♦ Output: identification of fault type
- Speech recognition
  - ♦ Input: spoken words
  - ♦ Output: identification of words written text
- Text to speech
  - ♦ Input: written text
  - ♦ Output: phonetic character string for pronunciation
- Sonar signal discrimination
  - ♦ Input: sonar responses
  - ♦ Output: classification of target
- Information encoding and compaction
  - ♦ Input: image or other pattern
  - ♦ Output: identical as input pattern
- Sequence recall
  - ♦ Input: current pattern in sequence
  - ♦ Output: next pattern in sequence

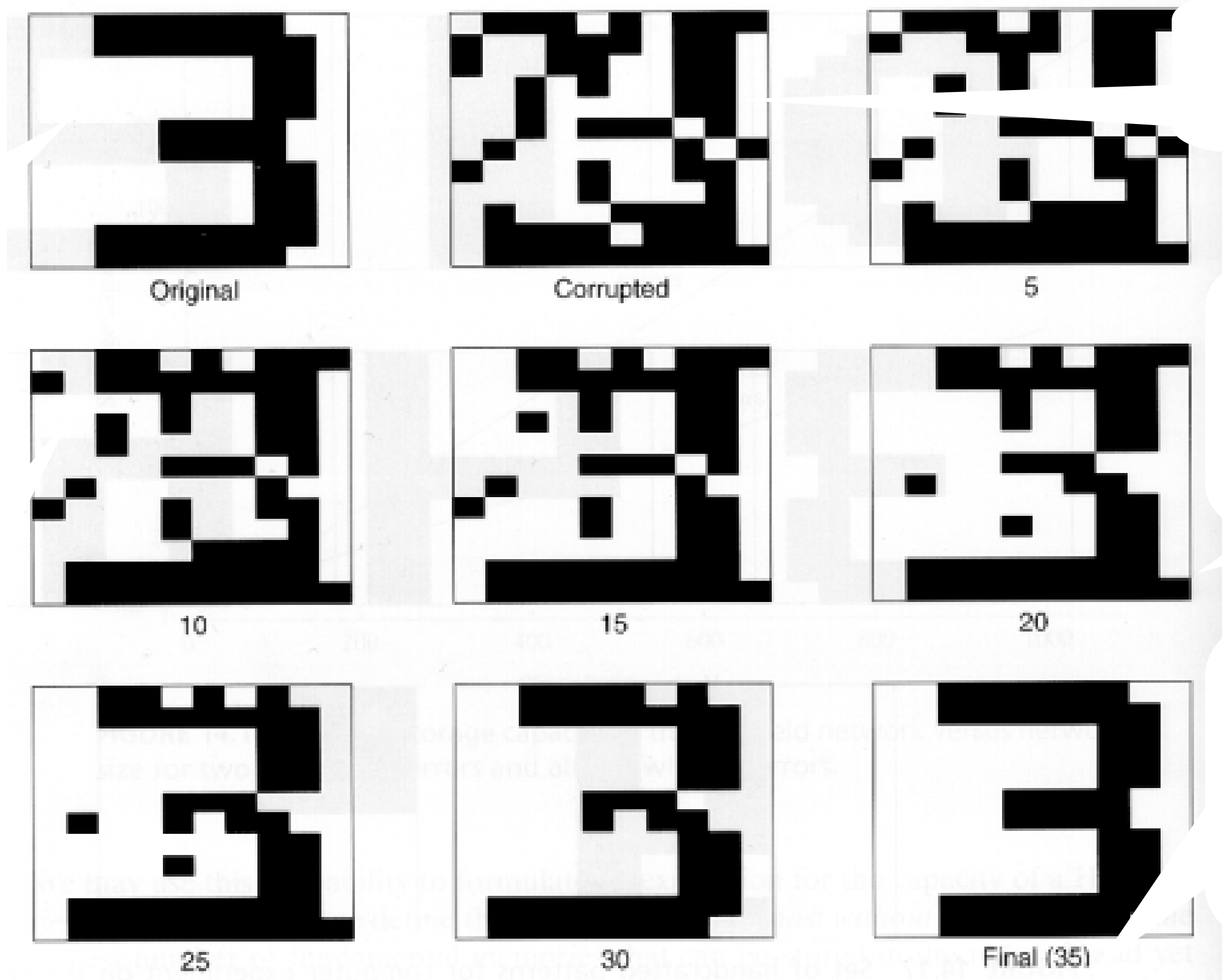
Application: Autonomous Land Vehicle (ALV)

- 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

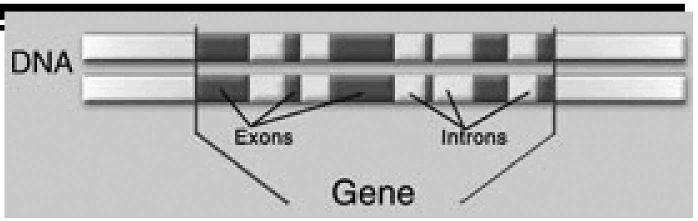


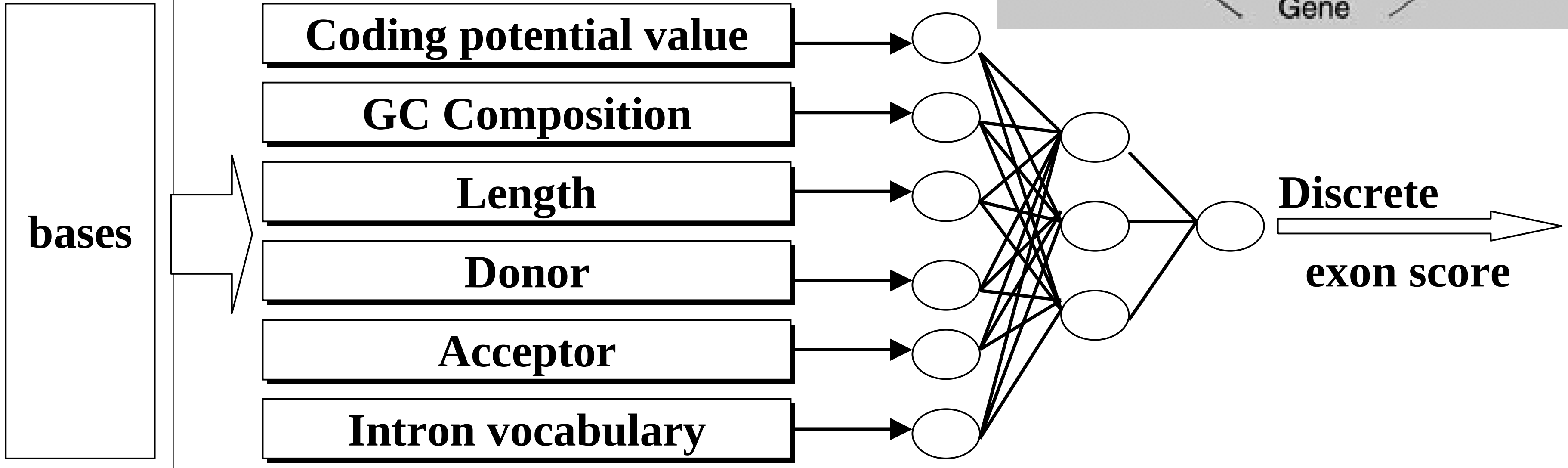


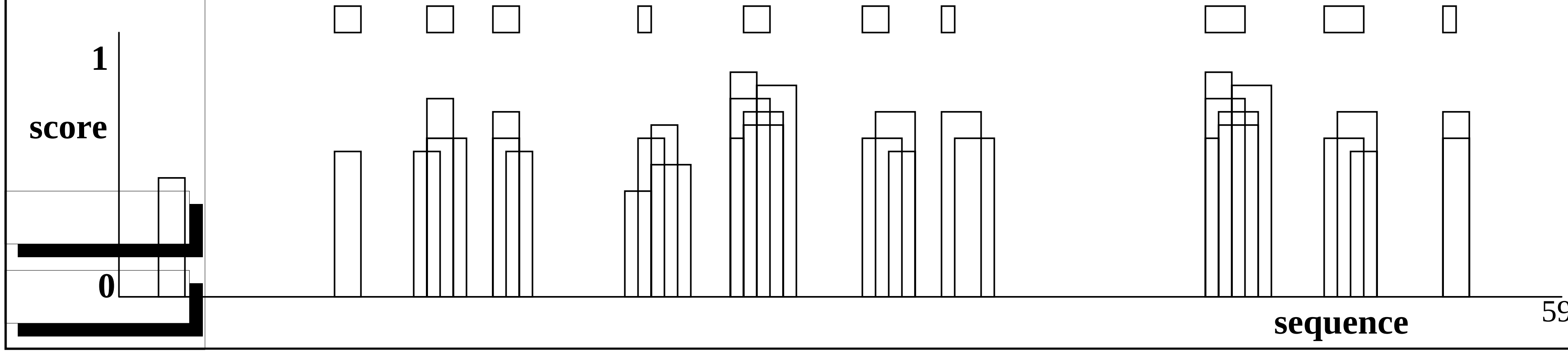
Application:
Data Recorrection by a Hopfield Network



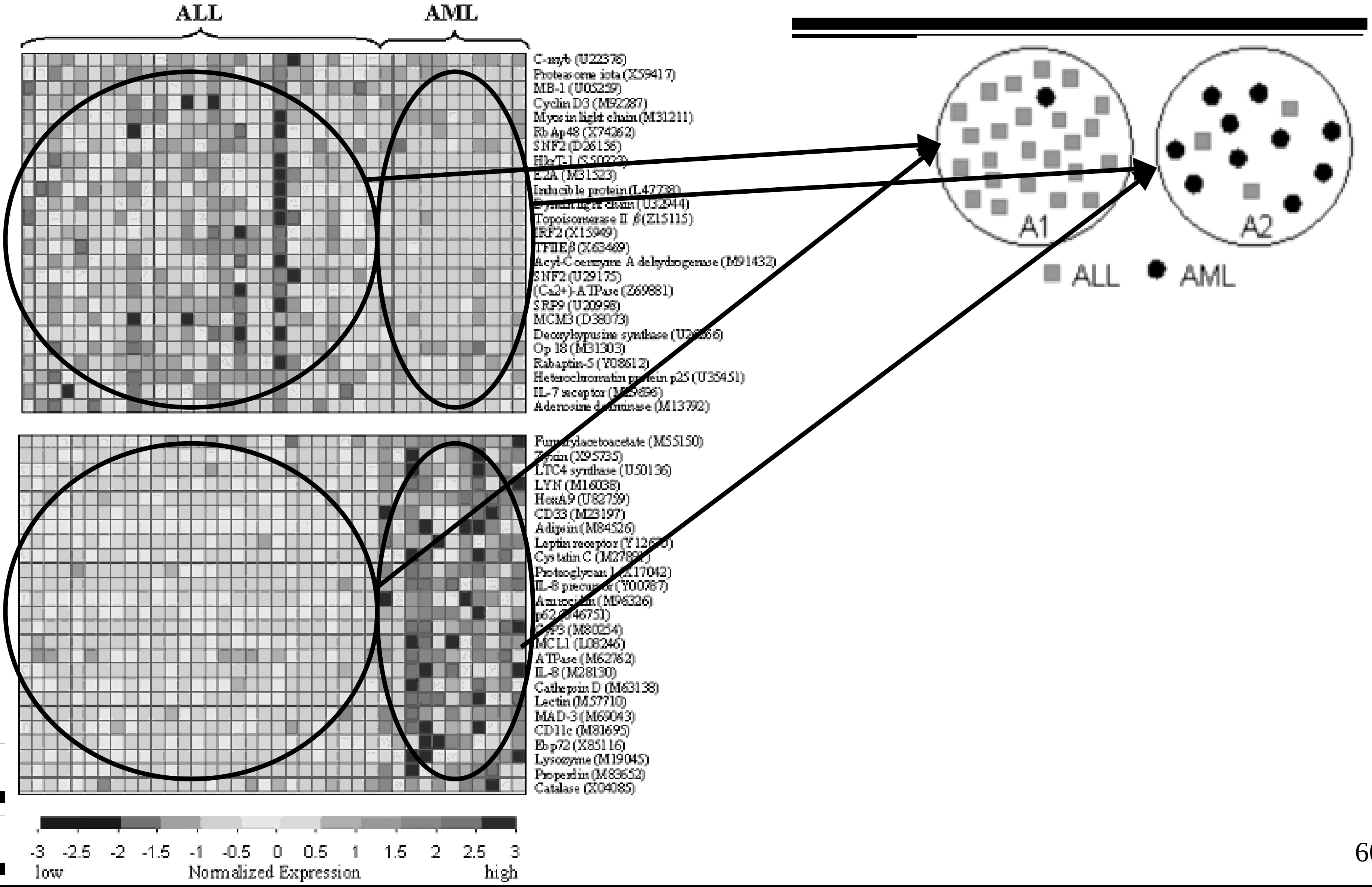
Application: Multilayer Perceptrons for Internal Exon Prediction: Bioinformatics



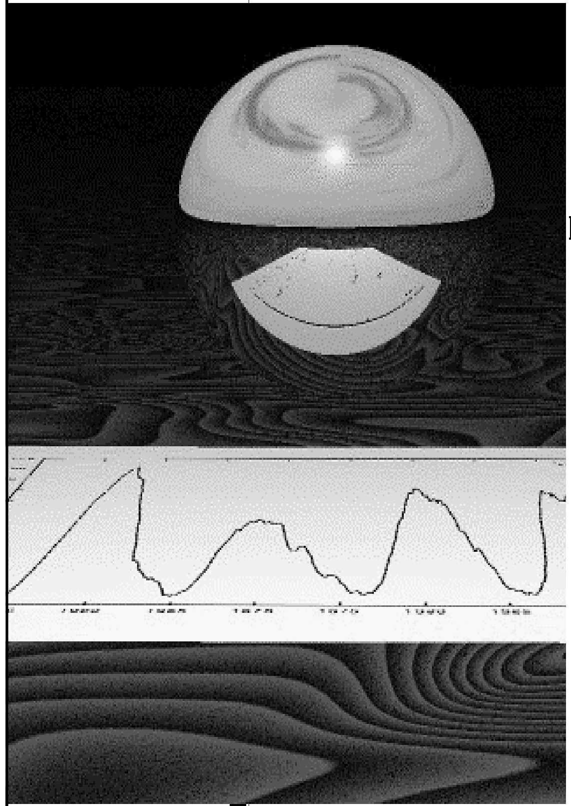


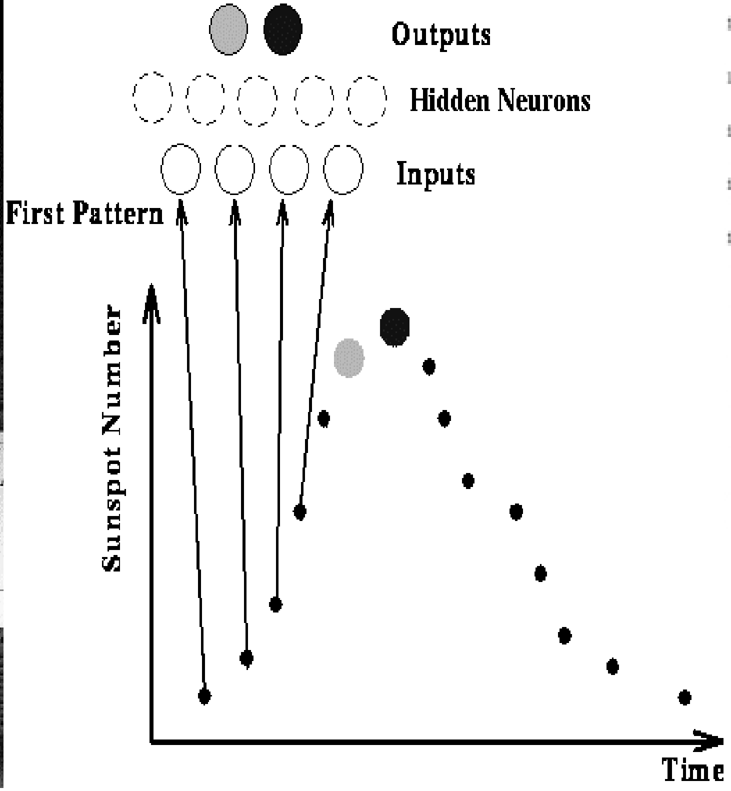


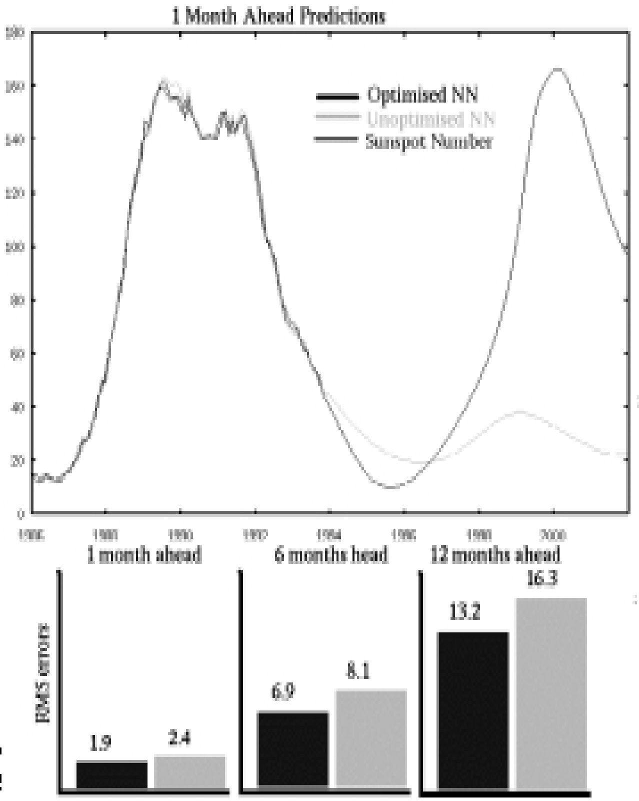
Application: Self-Organizing Maps for DNA Microarray Data Analysis



Application: Predicting the Sunspot Number with Neural Networks

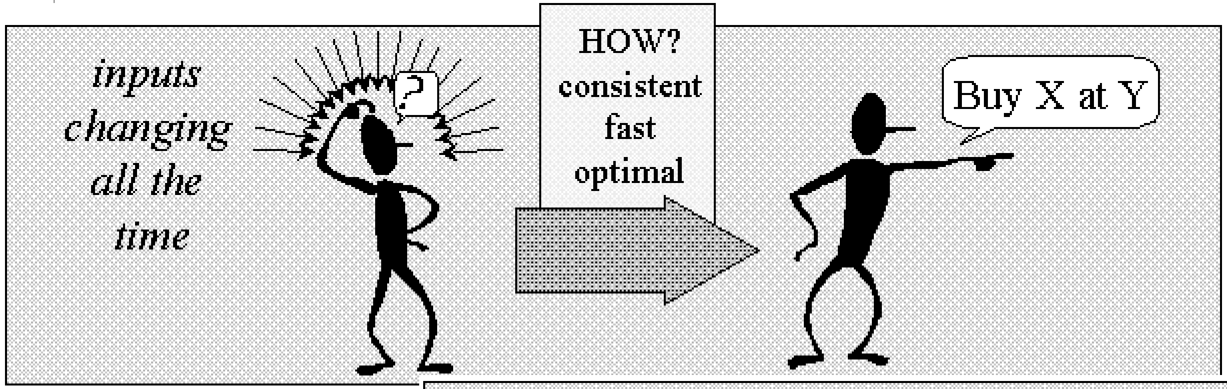


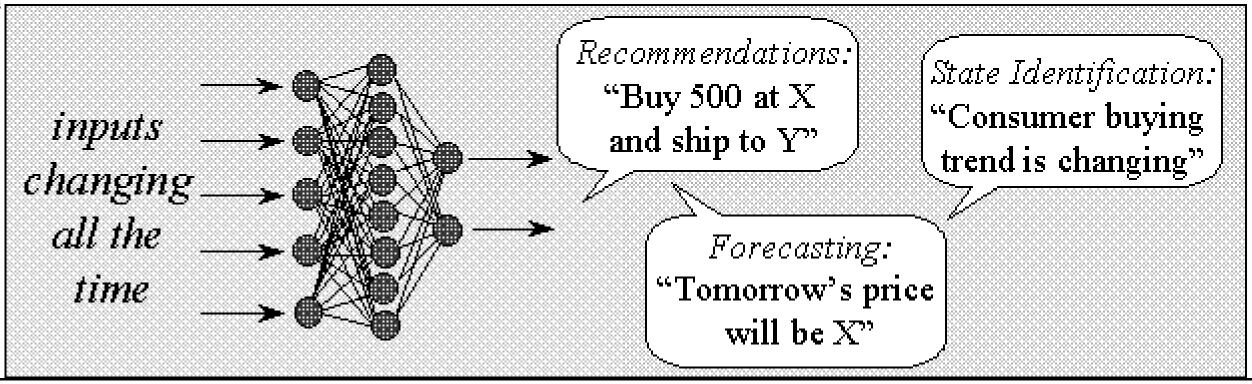




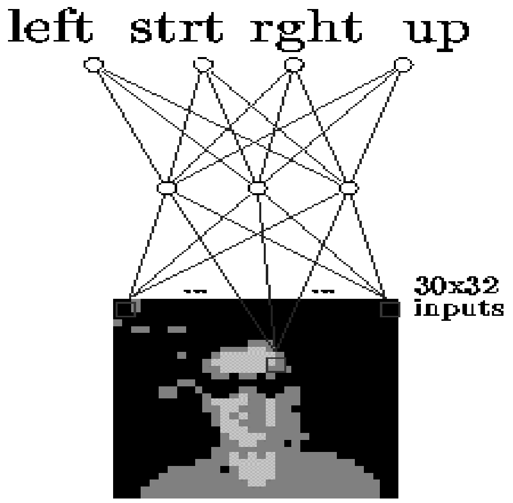
Application: Data Mining Techinques - Customer Relationship Management (CRM)

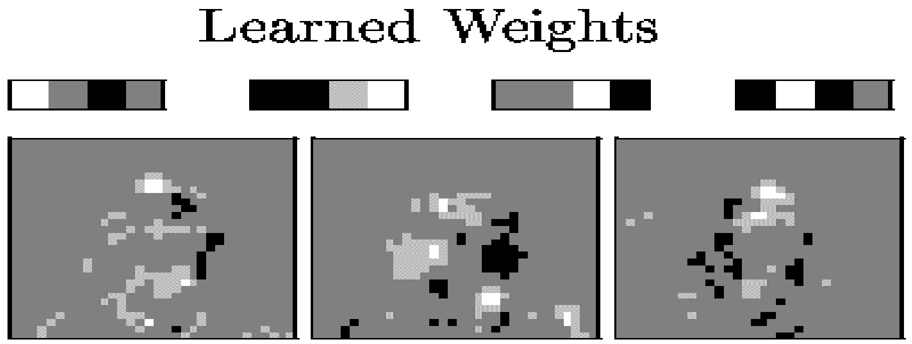
- Increased Customer Lifetime Value
- Increased Wallet Share
- Improved Customer Retention
- Segmentation of Customers by Profitability
- Segmentation of Customers by Risk of Default
- Integrating Data Mining into the Full Marketing Proce

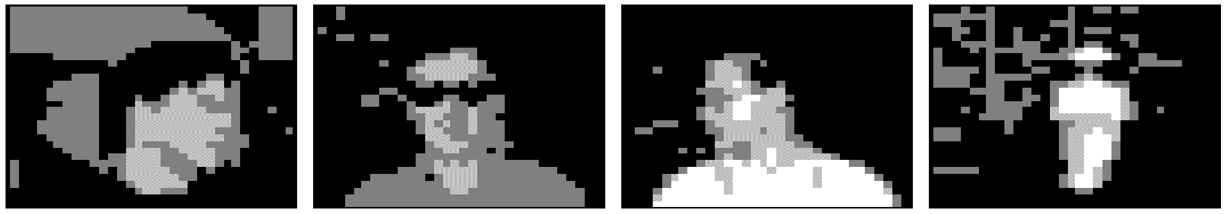



inputs changing all the time
Recommendations: "Buy 500 at X and ship to Y"
State Identification: "Consumer buying trend is changing"
Forecasting: "Tomorrow's price will be X"

Application: 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.

Face Recognition: Training Data

- 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.

Face Recognition: Factors for ANN Design

- Input encoding
- Output encoding
- Network graph structure
- Other learning algorithm parameters

Face Recognition: Input Coding

- 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.

Face Recognition: Output Coding

- 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

Face Recognition: Output Coding

- 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.

Face Recognition: Network Structure

- One hidden layer *vs.* 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 Sparc 5
  - Using 30 hidden units:
    - test accuracy for the face data = 91.5%
    - Training time = 1 hour on Sun Sparc 5

Face Recognition: Other Parameters

- 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

Summary and Further Information

Summary

- Neural networks are developed with the goal of modeling information processing and learning in the brain.
- Applied to a number of practical applications in various fields, including computational molecular biology.
- Bioinformatics can benefit much from neural networks.
- Can be viewed as a broad class of parameterized graphical models consisting of networks with interconnected.

Journals & Conferences

- Journals
  - IEEE Transactions on Neural Networks (*IEEE*)
  - Neural Networks (*Pergamon*)
  - Neural Computation (*MIT Press*)
  - Neurocomputing (*Elsevier Science Publishers*)
  - Neural Computing and Applications (*Springer-Verlag*)
  - Neural Processing Letters (*Kluwer*)

- Conferences
  - Neural Information Processing Systems *(NIPS)*
  - International Joint Conference on Neural Networks *(IJCNN)*
  - International Conference on Neural Information Processing (*ICONIP*)
  - International Conference on Artificial Neural Networks (*ICANN*)

WWW Resources

- FAQ file of the Usenet newsgroup comp.ai.neural-nets
  ftp://ftp.sas.com/pub/neural/FAQ.html
- Pacific Northwest National Laboratory (PNNL)
  http://www.emsl.pnl.gov:2080/proj/neuron/neural/
  newsgroups&mail lists, professional societies, conferences, journals, course, search
- Resources on Neural Networks
  http://www.it.uom.gr/pdp/DigitalLib/neural.htm
  software, people, papers, tutorial, FAQs
- Bibliographies on Neural Networks
  http://liinwww.ira.uka.de/bibliography/Neural/
- Connectionist Mailing List
  Subscription Address: Connectionists-Request@cs.cmu.edu
  Posting Address: Connectionists@cs.cmu.edu

Books

- Bishop, C. M., *Neural Networks for Pattern Recognition*, Oxford: Oxford University Press, 1995.
- Haykin S., *Neural Networks*, Macmillan College Publishing Company Inc., 1999.
- Hertz, J., Krogh, A., and Palmer, R., *Introduction to the Theory of Neural Computation*, Redwood City, CA: Addison-Wesley, 1991.
- Mitchell, T., *Machine Learning*, McGraw-Hill, 1997.
- Ripley, B. D, *Pattern Recognition and Neural Networks*, Cambridge: Cambridge University Press, 1996.
- Vladimir, C., and Filip, M., *Learning from Data*: *Concepts, Theory & Methods*, A Wiley-Interscience Publication, 1998.

For more information

http://scai.snu.ac.kr