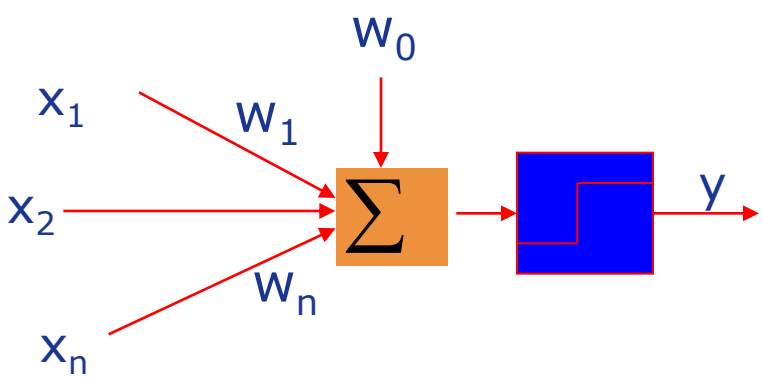
Machine Learning

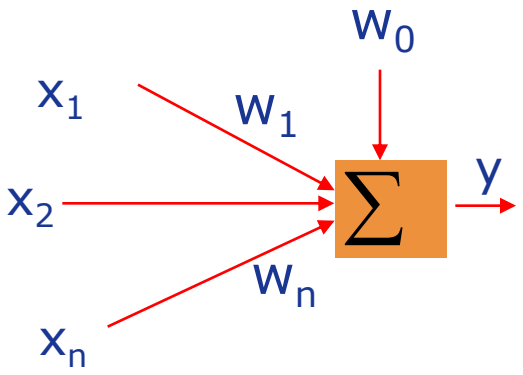
Lecture 4

Multilayer Perceptrons

Limitations of Single Layer Perceptron

- Only express linear decision surfaces





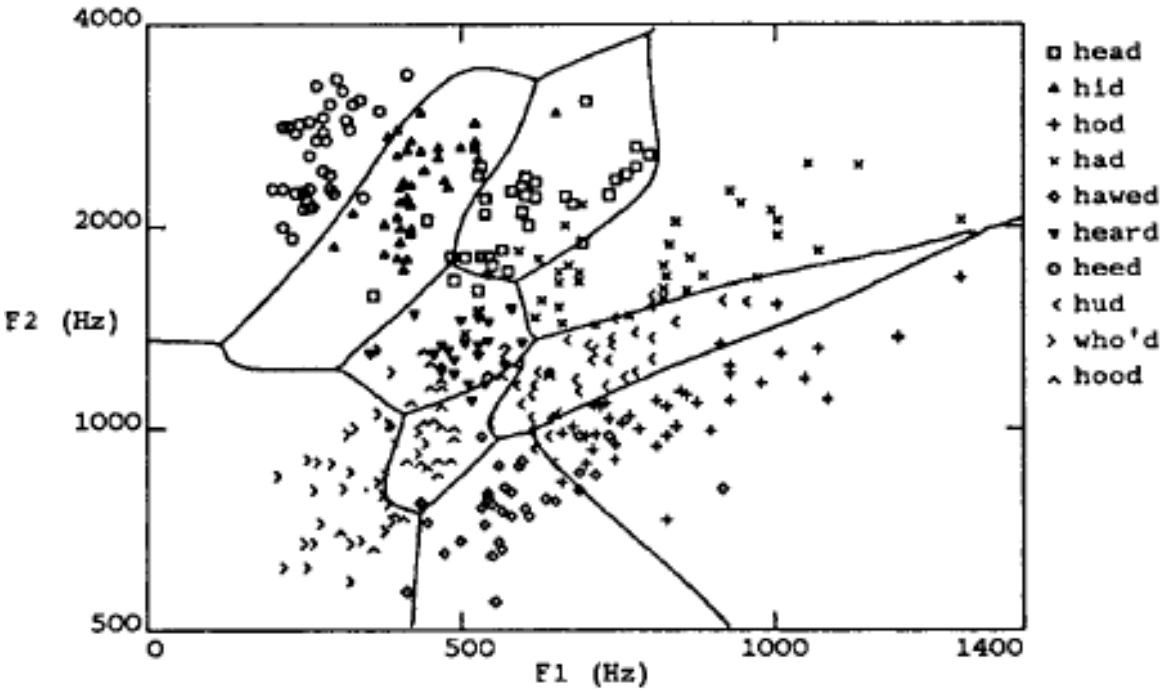
$$R = w_0 + \sum_{i=1}^{n} w_i x_i$$

$$Y = sign(R) = \begin{cases} +1; & \text{if} \quad R > 0 \\ -1, & \text{otherwise} \end{cases}$$

$$Y = w_0 + \sum_{i=1}^{n} w_i x_i$$

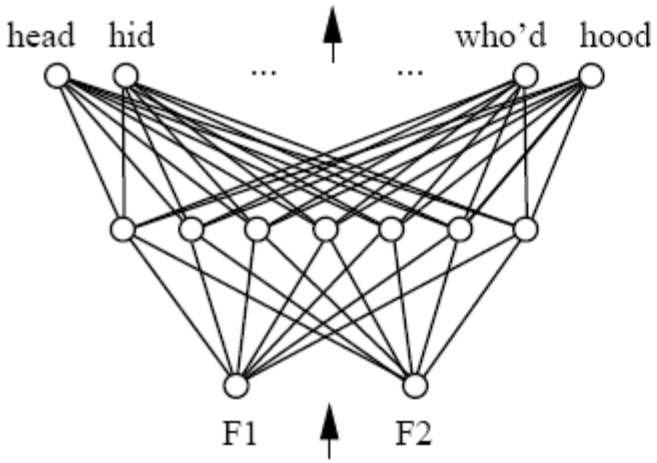
Nonlinear Decision Surfaces

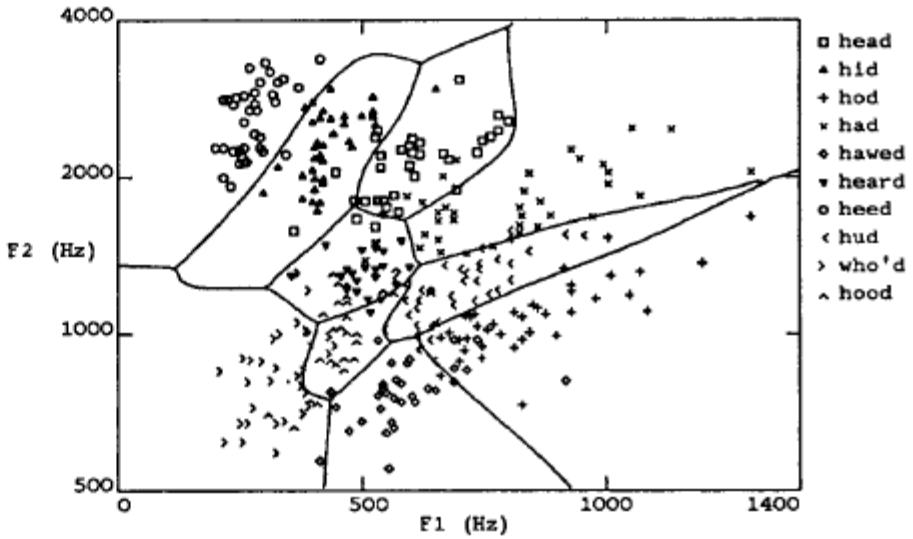
- A speech recognition task involves distinguishing 10 possible vowels all spoken in the context of 'h_d" (i.e., hit, had, head, etc). The input speech is represented by two numerical parameters obtained from spectral analysis of the sound, allowing easy visualization of the decision surfaces over the 2d feature space.



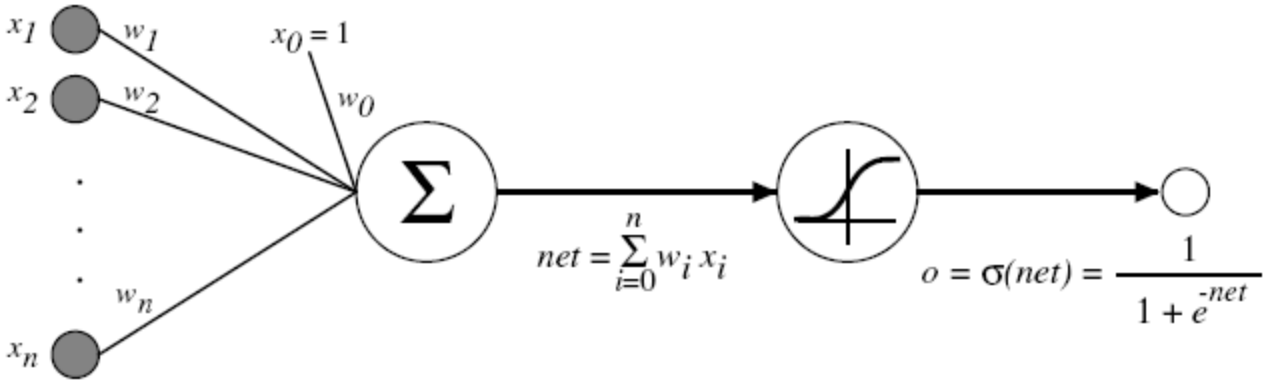
Multilayer Network

- We can build a multilayer network represent the highly nonlinear decision surfaces
- How?





Sigmoid Unit



$\sigma(x)$ is the sigmoid function

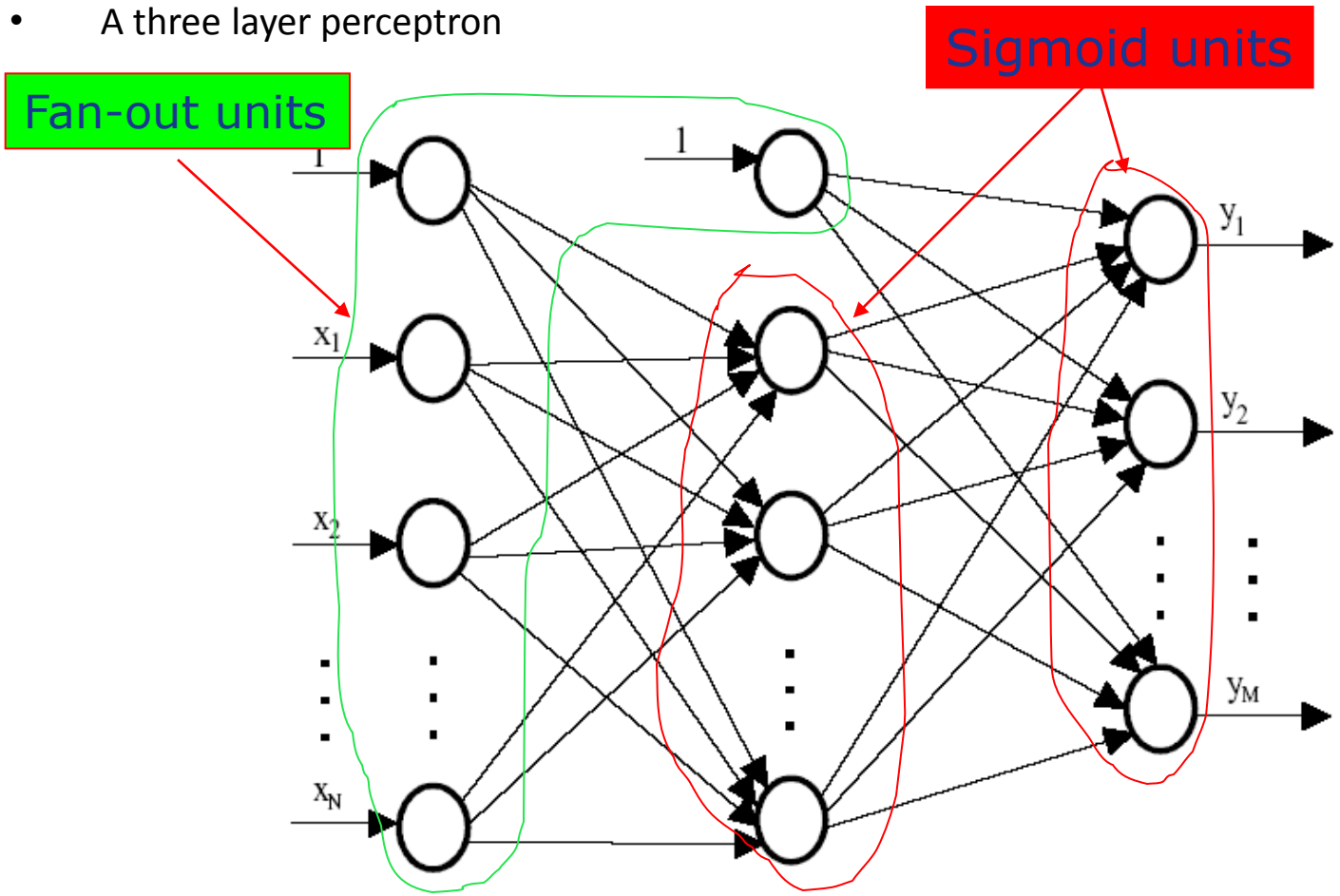
$$\frac{1}{1 + e^{-x}}$$

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

We can derive gradient decent rules to train

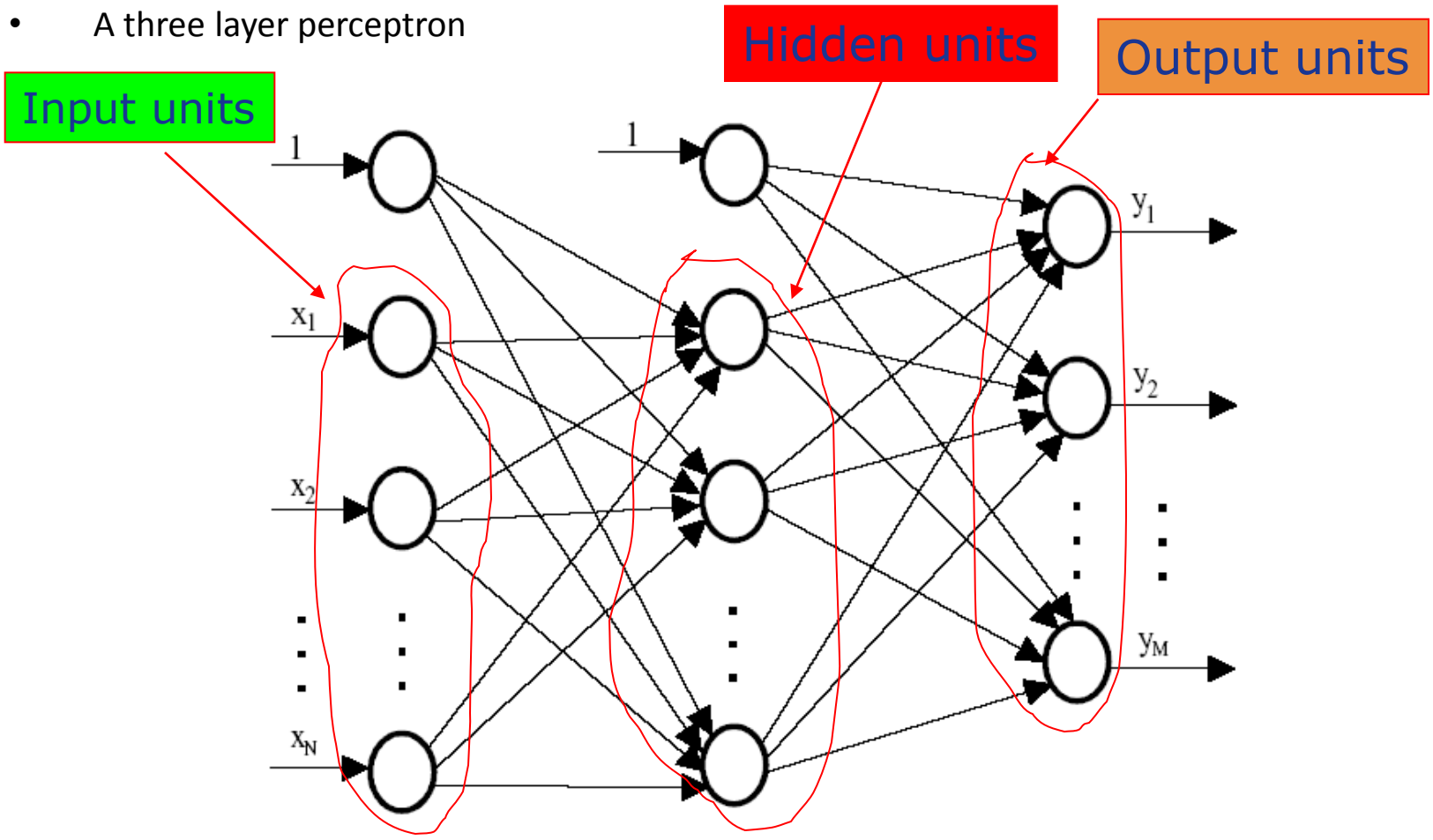
Multilayer Perceptron

- A three layer perceptron

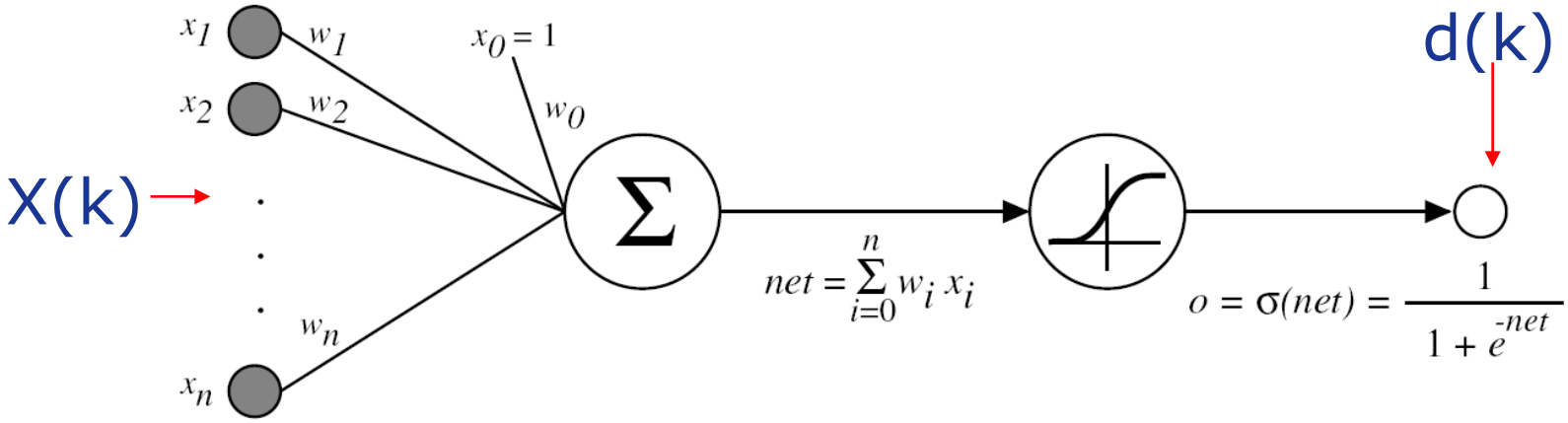


Multilayer Perceptron

- A three layer perceptron



Error Gradient for a Sigmoid Unit



$$E(W) \equiv \frac{1}{2} \sum_{k=1}^{K} (d(k) - y(k))^2 \longrightarrow y = o$$

Error Gradient for a Sigmoid Unit

$$E(W) \equiv \frac{1}{2}\sum_{k=1}^{K}(d(k)-y(k))^2$$

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_{k=1}^{K}(d(k)-y(k))^2\right) \\
&= \frac{1}{2}\sum_{k=1}^{K}\left(\frac{\partial}{\partial w_i}(d(k)-y(k))^2\right) \\
&= \frac{1}{2}\sum_{k=1}^{K}\left(2(d(k)-y(k))\frac{\partial}{\partial w_i}(d(k)-y(k))\right) \\
&= \sum_{k=1}^{K}\left((d(k)-y(k))\frac{\partial}{\partial w_i}(-y(k))\right) \\
&= -\sum_{k=1}^{K}\left((d(k)-y(k))\frac{\partial y(k)}{\partial net(k)}\frac{\partial net(k)}{\partial w_i}\right)
\end{aligned}$$

Error Gradient for a Sigmoid Unit

$$\frac{\partial y(k)}{\partial net(k)} = \frac{\partial \sigma(net(k))}{\partial net(k)} = \sigma(net(k))(1-\sigma(net(k))) = y(k)(1-y(k))$$

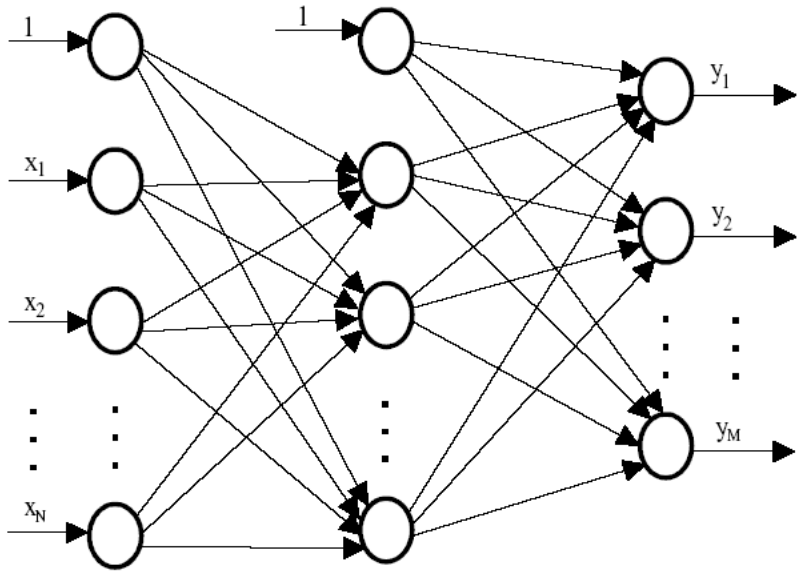
$$\frac{\partial net(k)}{\partial w_i} = \frac{\partial (X(k) \bullet W)}{\partial w_i} = x_i(k)$$

$$\frac{\partial E}{\partial w_i} = -\sum_{k=1}^{K} \left((d(k)-y(k)) \frac{\partial y(k)}{\partial net(k)} \frac{\partial net(k)}{\partial w_i} \right)$$

$$= -\sum_{k=1}^{K} ((d(k)-y(k))y(k)(1-y(k))x_i(k))$$

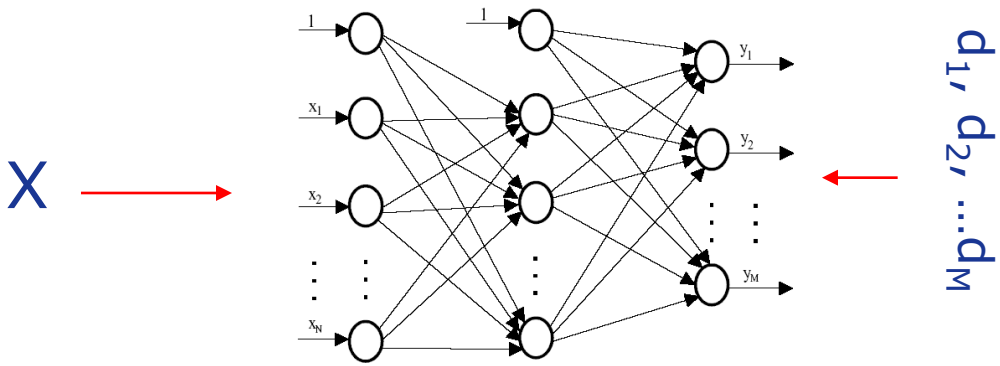
Back-propagation Algorithm

- For training multilayer perceptrons



Back-propagation Algorithm

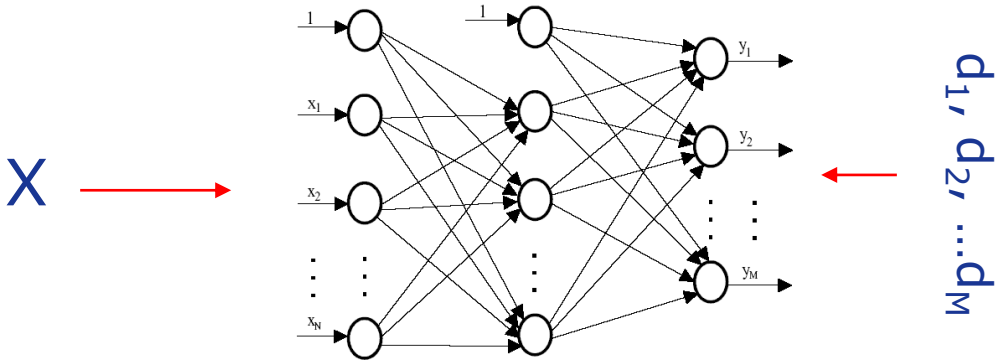
- For each training example, training involves following steps



Step 1: Present the training sample, calculate the outputs

Back-propagation Algorithm

- For each training example, training involves following steps

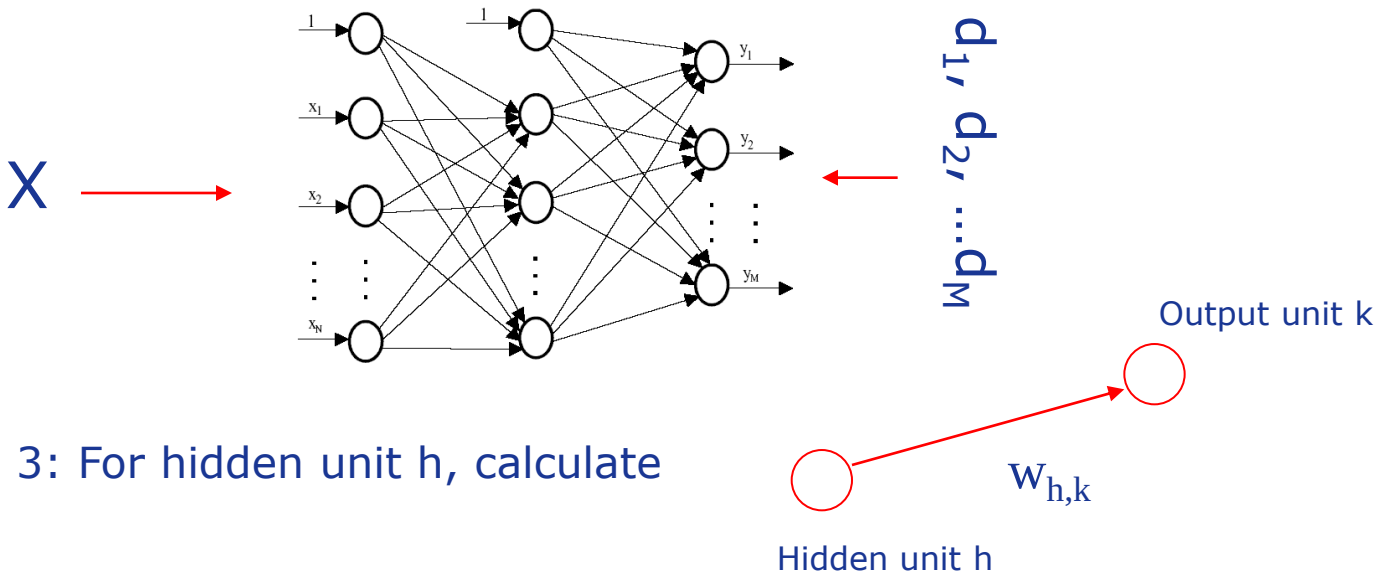


Step 2: For each output unit k, calculate

$$\delta_k = y_k(1-y_k)(d_k - y_k)$$

Back-propagation Algorithm

- For each training example, training involves following steps



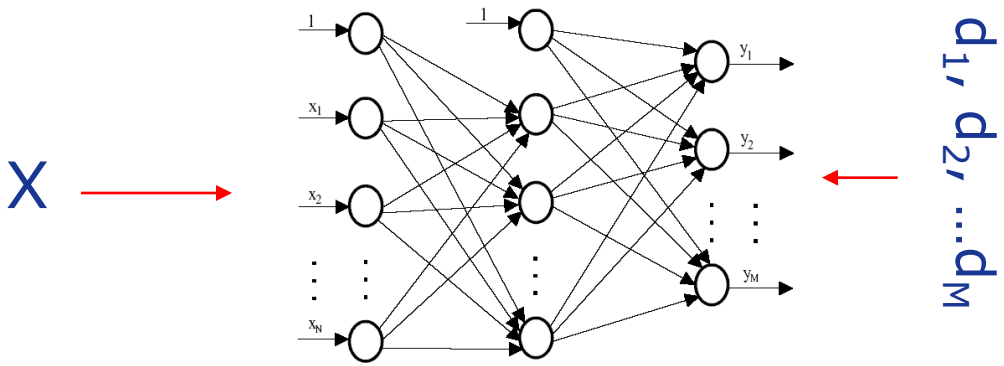
Step 3: For hidden unit h, calculate

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

Error back-propagation

Back-propagation Algorithm

- For each training example, training involves following steps

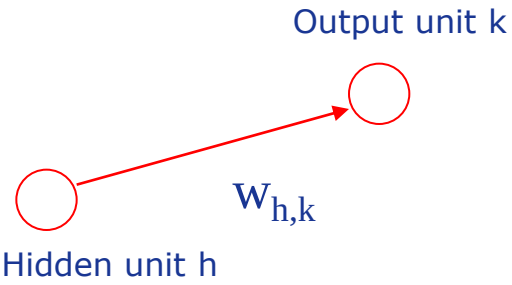


Step 4: Update the output layer weights, $w_{h,k}$

$$w_{h,k} \leftarrow w_{h,k} + \Delta w_{h,k}$$

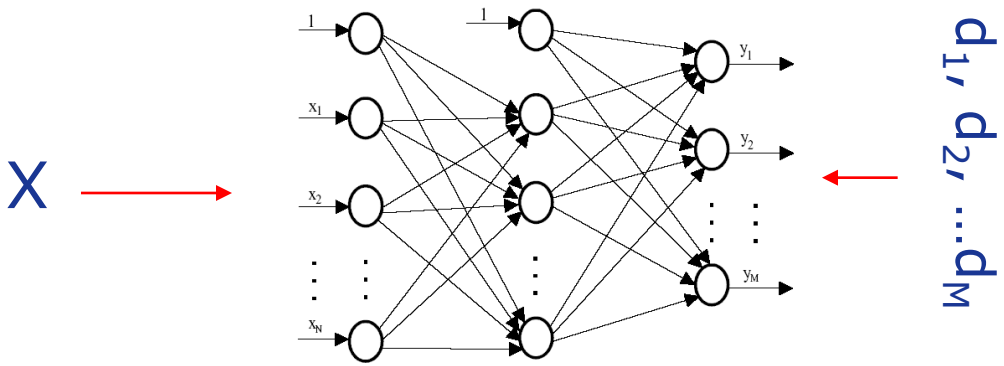
$$\Delta w_{h,k} = \eta \delta_k o_h$$

where o_h is the output of hidden layer h



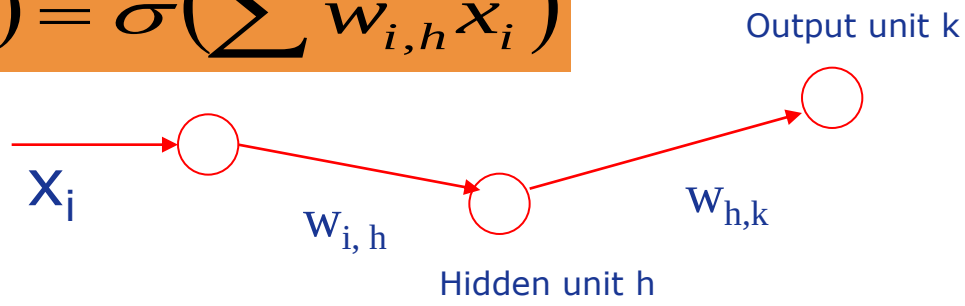
Back-propagation Algorithm

- For each training example, training involves following steps



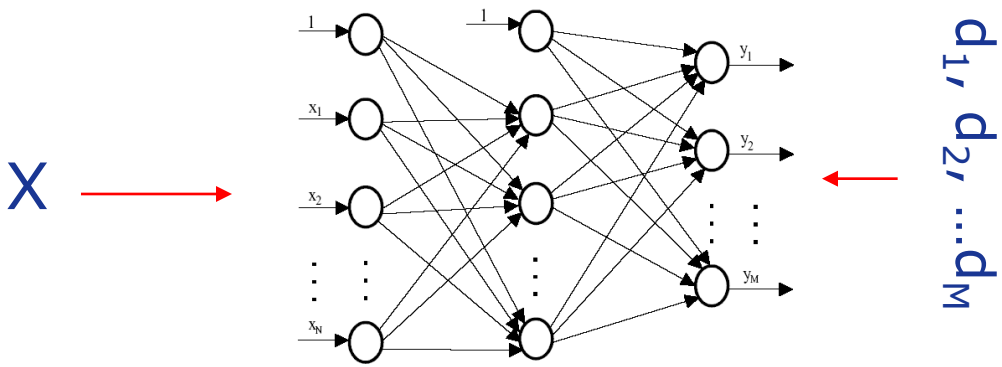
o_h is the output of hidden unit h

$$o_h = \sigma(net_h) = \sigma\left(\sum w_{i,h} x_i\right)$$



Back-propagation Algorithm

- For each training example, training involves following steps



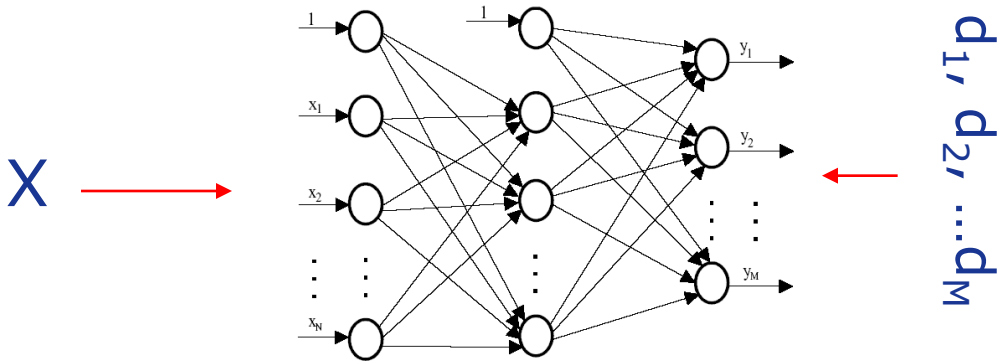
Step 4: Update the output layer weights, $w_{h,k}$

$$w_{h,k} \leftarrow w_{h,k} + \Delta w_{h,k}$$

$$\Delta w_{h,k} = \eta y_k (1 - y_k)(d_k - y_k)\sigma\left(\sum w_{i,h} x_i\right)$$

Back-propagation Algorithm

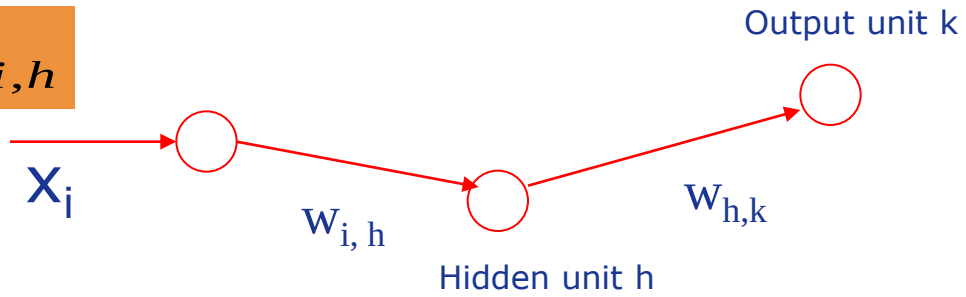
- For each training example, training involves following steps



Step 5: Update the hidden layer weights, $w_{i,h}$

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

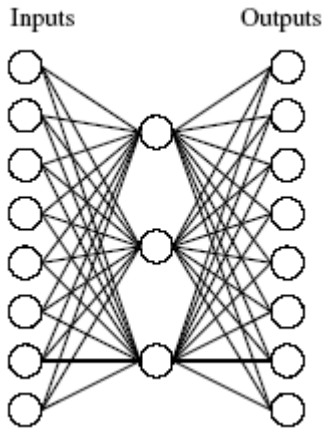
$$\Delta w_{i,h} = \eta \delta_h x_i$$



Back-propagation Algorithm

- Gradient descent over entire network weight vector
- Will find a local, not necessarily a global error minimum.
- In practice, it often works well (can run multiple times)
- Minimizes error over all training samples
  - Will it generalize will to subsequent examples? i.e., will the trained network perform well on data outside the training sample
- Training can take thousands of iterations
- After training, use the network is fast

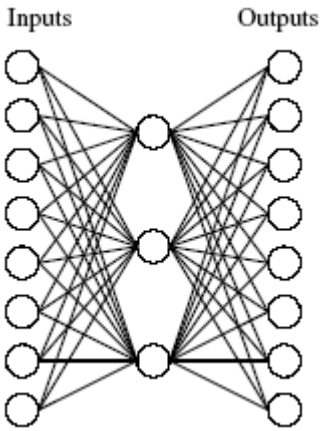
Learning Hidden Layer Representation



Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

Can this be learned?

Learning Hidden Layer Representation

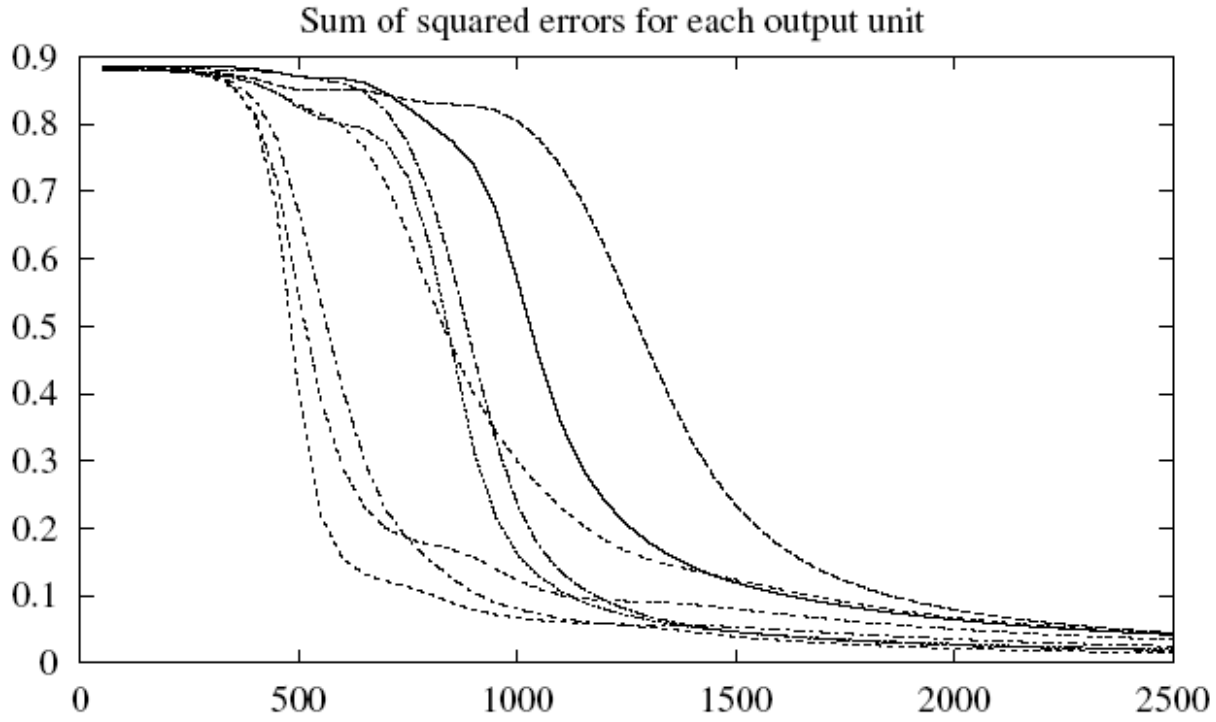


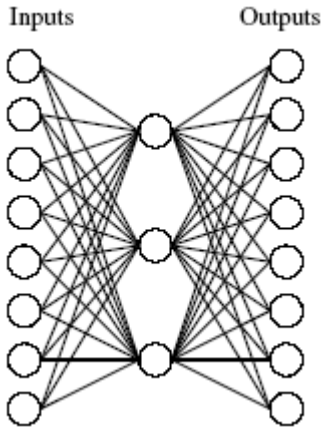
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.01	.11	.88	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.22	.99	.99	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

Learned hidden layer representation

Learning Hidden Layer Representation

- Training

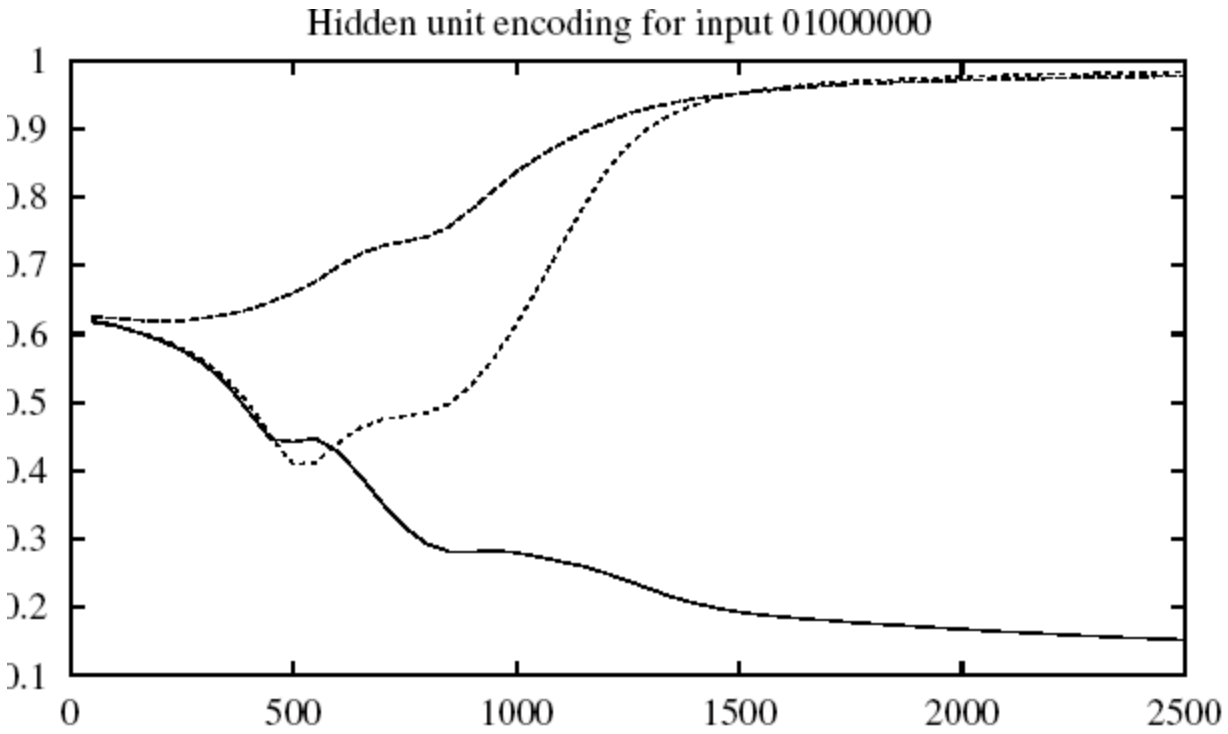


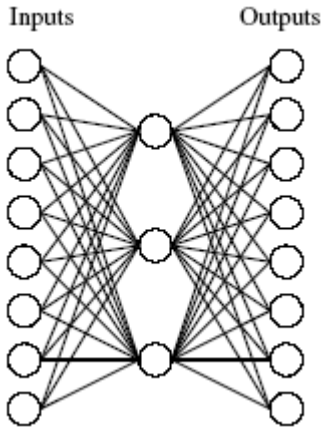


The evolving sum of squared errors for each of the eight output units

Learning Hidden Layer Representation

- Training





The evolving hidden layer representation for the input "01000000"

Expressive Capabilities

Boolean functions:

- Every boolean function can be represented by network with single hidden layer
- but might require exponential (in number of inputs) hidden units

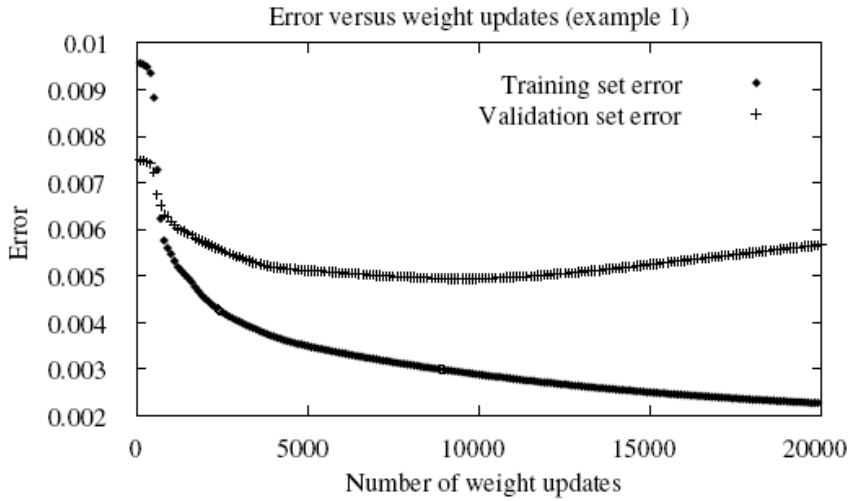
Continuous functions:

- Every bounded continuous function can be approximated with arbitrarily small error, by network with one hidden layer [Cybenko 1989; Hornik et al. 1989]
- Any function can be approximated to arbitrary accuracy by a network with two hidden layers [Cybenko 1988].

Generalization, Overfitting and Stopping Criterion

- What is the appropriate condition for stopping weight update loop?
  - Continue until the error E falls below some predefined value
    - Not a very good idea – Back-propagation is susceptible to overfitting the training example at the cost of decreasing generalization accuracy over other unseen examples

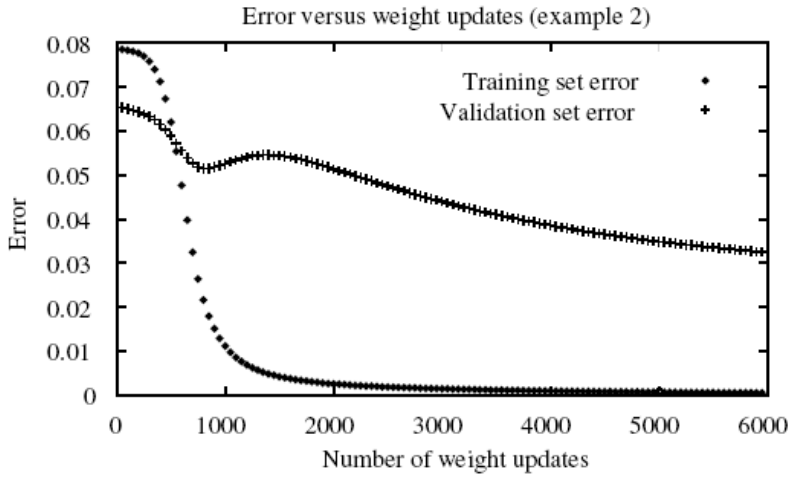
Generalization, Overfitting and Stopping Criterion



A training set

A validation set

Stop training when the validation set has the lowest error



Application Examples

- NETtalk (http://www.cnl.salk.edu/ParallelNetsPronounce/index.php)

- Training a network to pronounce English text

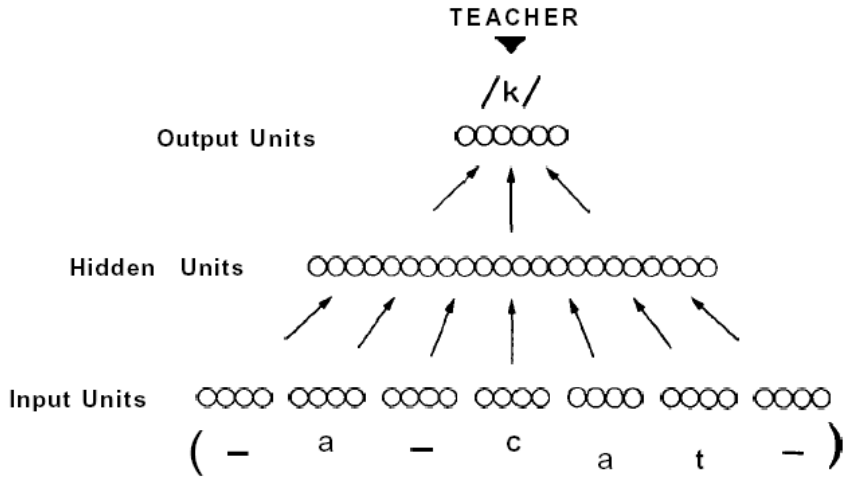


Figure 1: Schematic drawing of the NETtalk network architecture. A window of letters in an English text is fed to an array of 203 input units. Information from these units is transformed by an intermediate layer of 80 'hidden" units to produce patterns of activity in 26 output units. The connections in the network are specified by a total of 18629 weight parameters (including a variable threshold for each unit).

Application Examples

- NETtalk (http://www.cnl.salk.edu/ParallelNetsPronounce/index.php)
- Training a network to pronounce English text
  - The input to the network: 7 consecutive characters from some written text, presented in a moving windows that gradually scanned the text
  - The desired output: A phoneme code which could be directed to a speech generator, given the pronunciation of the letter at the centre of the input window
  - The architecture: 7x29 inputs encoding 7 characters (including punctuation), 80 hidden units and 26 output units encoding phonemes.

Application Examples

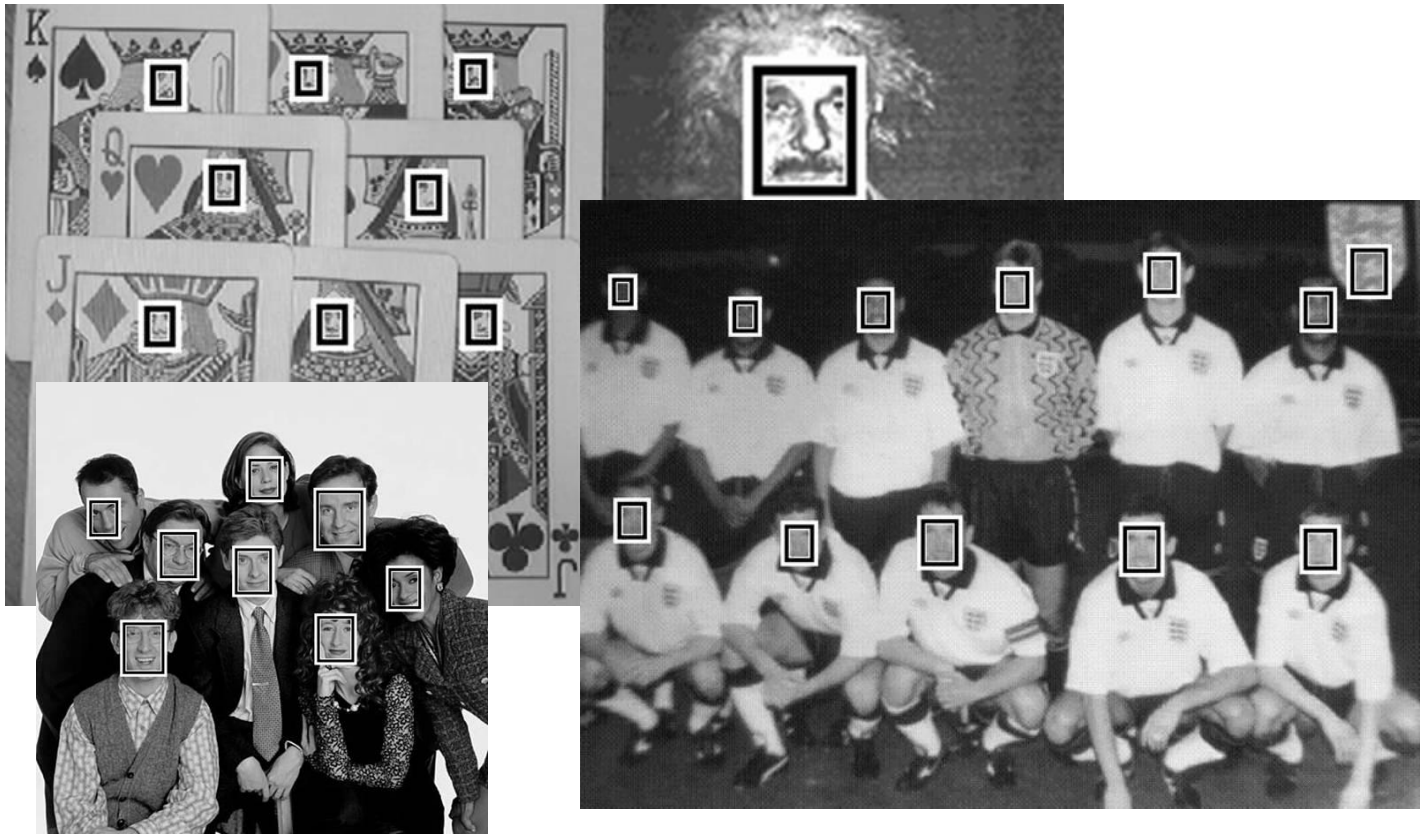
- NETtalk (http://www.cnl.salk.edu/ParallelNetsPronounce/index.php)
- Training a network to pronounce English text
  - Training examples: 1024 words from a side-by-side English/phoneme source
  - After 10 epochs, intelligible speech
  - After 50 epochs, 95% accuracy
  - It first learned gross features such as the division points between words and gradually refines its discrimination, sounding rather like a child learning to talk

Application Examples

- NETtalk (http://www.cnl.salk.edu/ParallelNetsPronounce/index.php)
- Training a network to pronounce English text
  - Internal Representation: Some internal units were found to be representing meaningful properties of the input, such as the distinction between vowels and consonants.
  - Testing: After training, the network was tested on a continuation of the side-by-side source, and achieved 78% accuracy on this ***generalization*** task, producing quite intelligible speech.
  - Damaging the network by adding random noise to the connection weights, or by removing some units, was found to degrade performance continuously (not catastrophically as expected for a digital computer), with a rather rapid recovery after retraining.

Application Examples

- Neural Network-based Face Detection

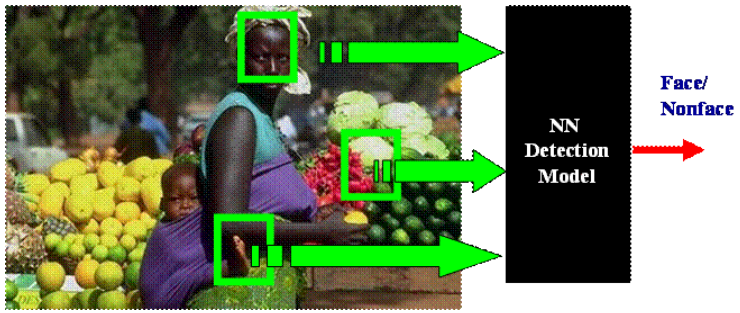


Application Examples

- Neural Network-based Face Detection

Application Examples

- Neural Network-based Face Detection
  - It takes 20 x 20 pixel window, feeds it into a NN, which outputs a value ranging from –1 to +1 signifying the presence or absence of a face in the region
  - The window is applied at every location of the image
  - To detect faces larger than 20 x 20 pixel, the image is repeatedly reduced in size



Application Examples

- Neural Network-based Face Detection (http://www.ri.cmu.edu/projects/project_271.html)

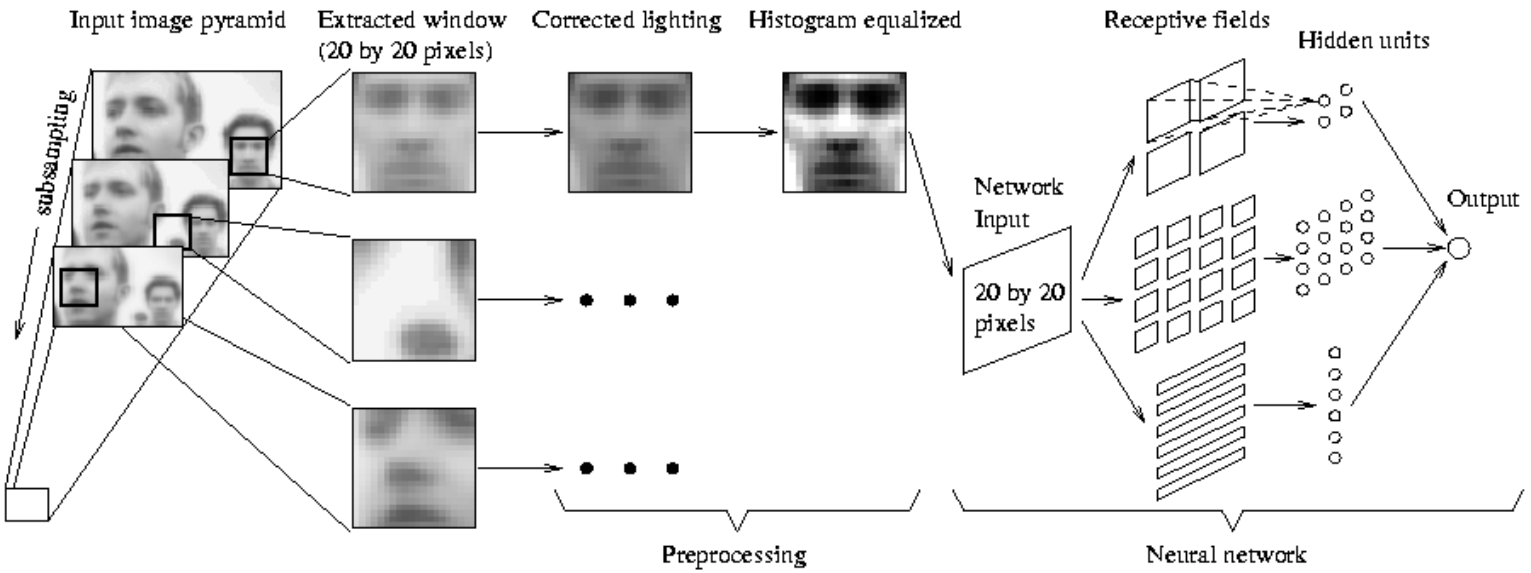


Figure 1: The basic algorithm used for face detection.

Application Examples

- Neural Network-based Face Detection (http://www.ri.cmu.edu/projects/project_271.html)

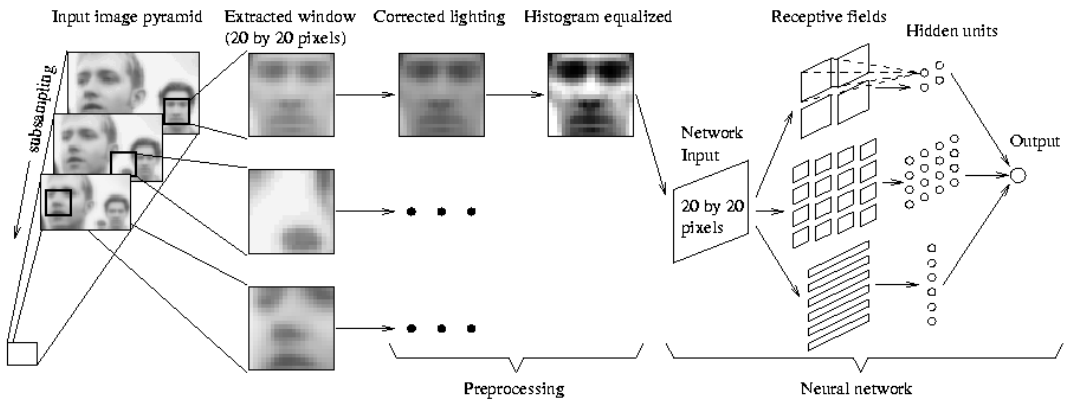


Figure 1: The basic algorithm used for face detection.

- Three-layer feedforward neural networks
- Three types of hidden neurons
  - 4 look at 10 x 10 subregions
  - 16 look at 5x5 subregions
  - 6 look at 20x5 horizontal stripes of pixels

Application Examples

- Neural Network-based Face Detection (http://www.ri.cmu.edu/projects/project_271.html)

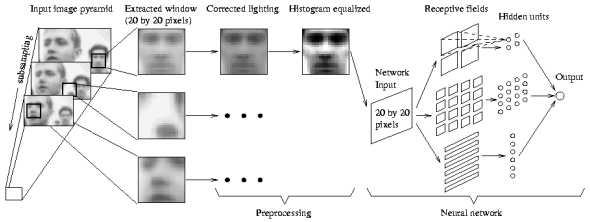


Figure 1: The basic algorithm used for face detection.

  - Training samples
  - 1050 initial face images. More face example are generated from this set by rotation and scaling. Desired output +1
  - Non-face training samples: Use a bootstrappng technique to collect 8000 non-face training samples from 146,212,178 subimage regions! Desired output -1

Application Examples

- Neural Network-based Face Detection (http://www.ri.cmu.edu/projects/project_271.html)

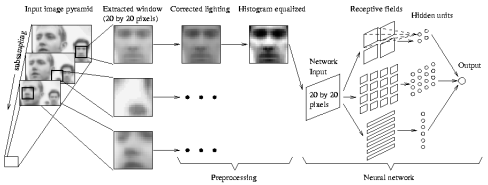
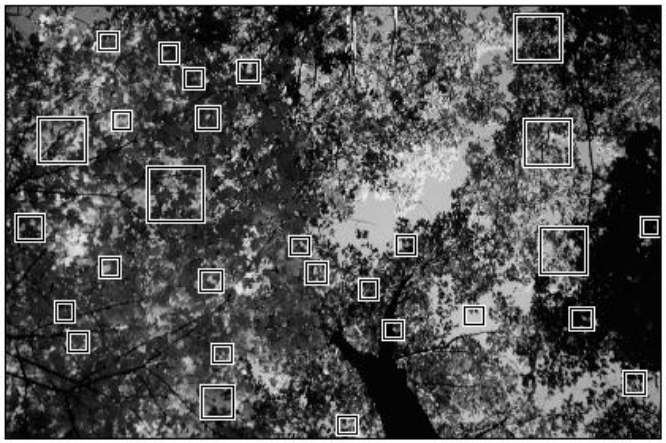
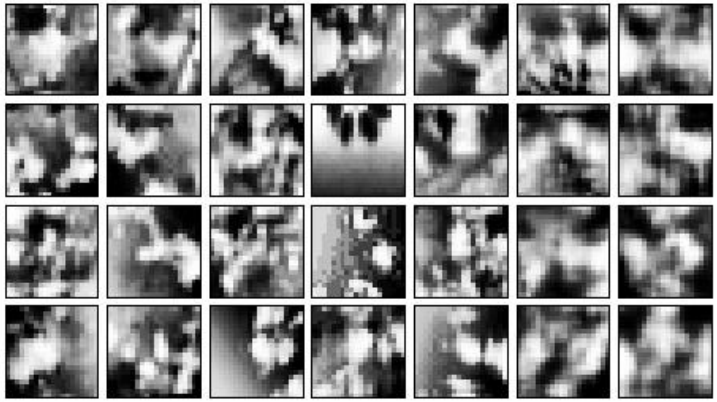


Figure 1: The basic algorithm used for face detection.

- Training samples: Non-face training samples





Application Examples

- Neural Network-based Face Detection (http://www.ri.cmu.edu/projects/project_271.html)

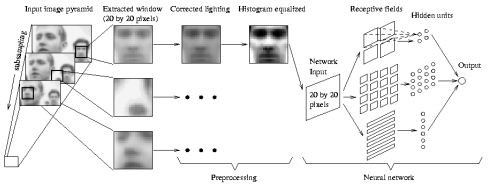
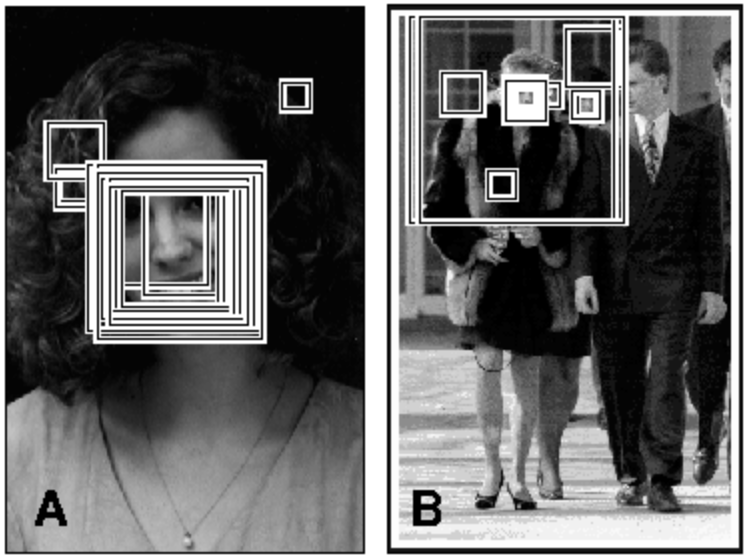


Figure 1: The basic algorithm used for face detection.

- Post-processing and face detection



Application Examples

- Neural Network-based Face Detection (http://www.ri.cmu.edu/projects/project_271.html)

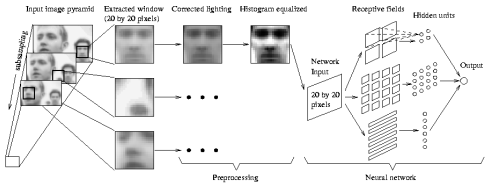


Figure 1: The basic algorithm used for face detection.

- Results and Issues

- 77.% ~ 90.3% detection rate (130 test images)
- Process 320x240 image in 2 – 4 seconds on a 200MHz R4400 SGI Indigo 2

Further Readings

1. T. M. Mitchell, Machine Learning, McGraw-Hill International Edition, 1997

Chapter 4

Tutorial/Exercise Question

1. Assume that a system uses a three-layer perceptron neural network to recognize 10 hand-written digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Each digit is represented by a 9 x 9 pixels binary image and therefore each sample is represented by an 81-dimensional binary vector. The network uses 10 neurons in the output layer. Each of the output neurons signifies one of the digits. The network uses 120 hidden neurons. Each hidden neuron and output neuron also has a bias input.

   (i) How many connection weights does the network contain?

   (ii) For the training samples from each of the 10 digits, write down their possible corresponding desired output vectors.

   (iii) Describe briefly how the backprogation algorithm can be applied to train the network.

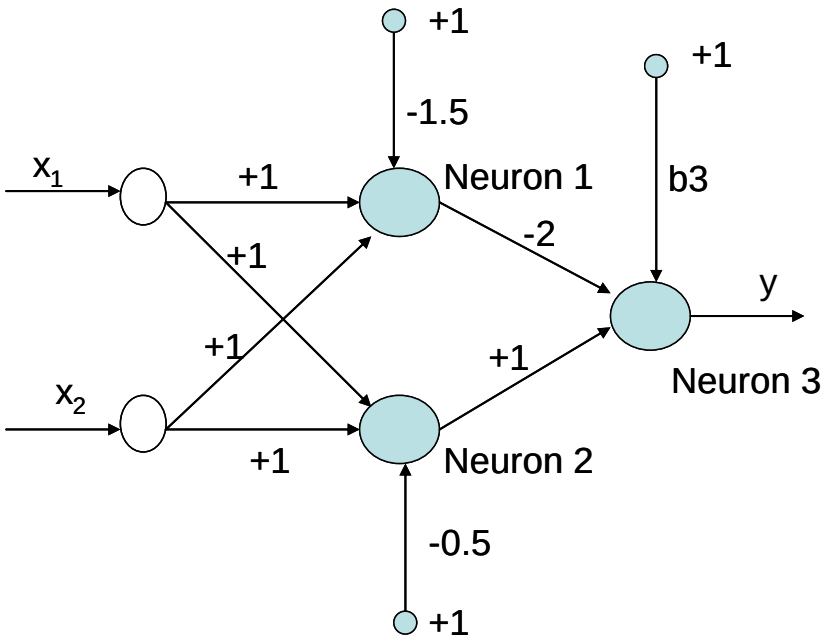
   (iv) Describe briefly how a trained network will be applied to recognize an unknown input.

Tutorial/Exercise Question

2. The network shown in the Figure is a 3 layer feed forward network. Neuron 1, Neuron 2 and Neuron 3 are McCulloch-Pitts neurons which use a threshold function for their activation function. All the connection weights, the bias of Neuron 1 and Neuron 2 are shown in the Figure. Find an appropriate value for the bias of Neuron 3, **b3**, to enable the network to solve the XOR problem (assume bits 0 and 1 are represented by level 0 and +1, respectively). Show your working process.

XOR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Tutorial/Exercise Question

3. Consider a 3 layer perceptron with two inputs a and b, one hidden unit c and one output unit d. The network has five weights which are initialized to have a value of 0.1. Give their values after the presentation of each of the following training samples

Input		Desired Output
a=1	b=0	1
b=0	b=1	0