University of Nottingham
UK | CHINA | MALAYSIA

COMP3055 Machine Learning

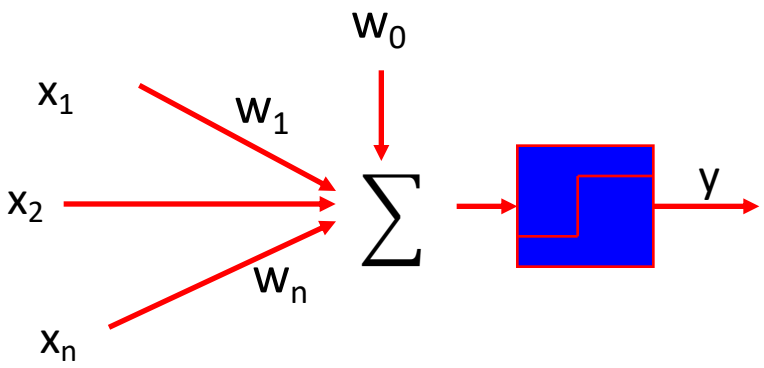
Topic 4 – Multilayer Perceptrons

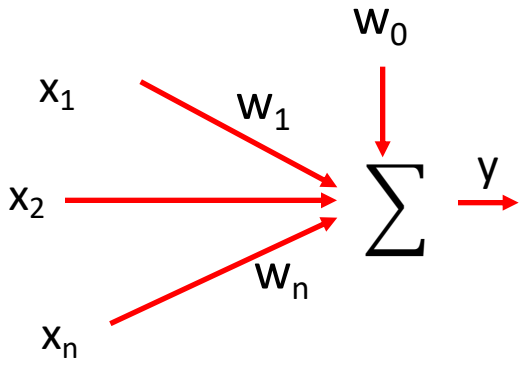
Dr. Zheng LU

2018 Autumn

Limitations of Single Layer Perceptron

Only express linear decision surfaces





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

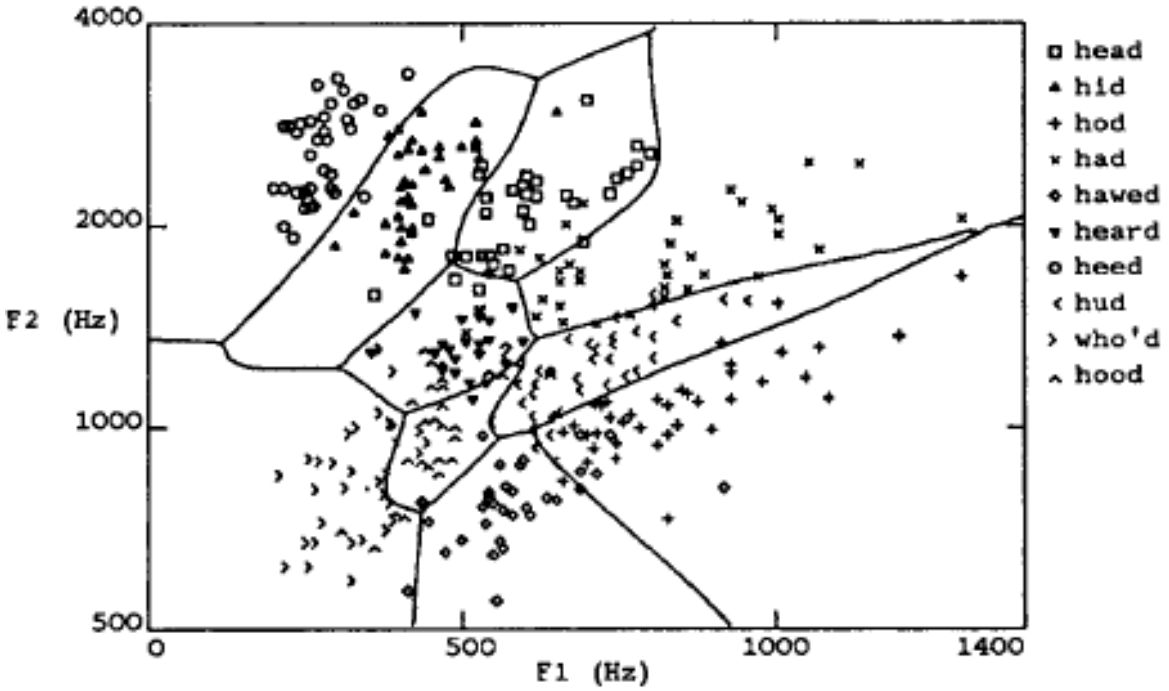
$$o = sign(R) = \begin{cases} +1, & if\ R > 0 \\ -1, & otherwise \end{cases}$$

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

Nonlinear Decision Surfaces

A Speech Recognition Example:

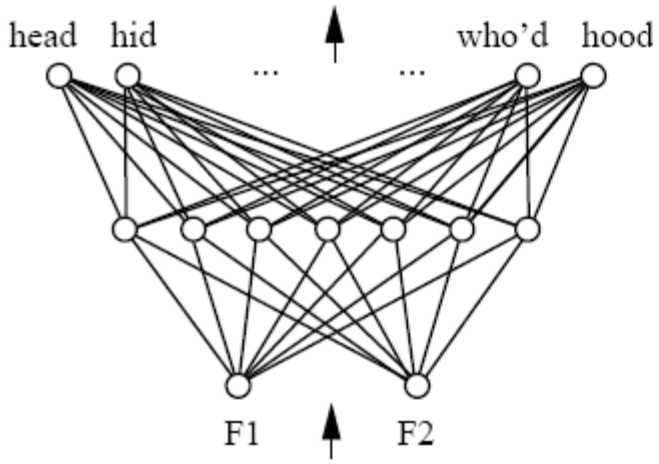
- Problem: recognize 10 possible vowels
- Dataset: pronunciation of words starting ‘h’ sound and ending with ‘d‘ sound (i.e., hit, had, head, etc).
- Features: two numerical numbers representing frequencies, obtained from spectral analysis of the sound.

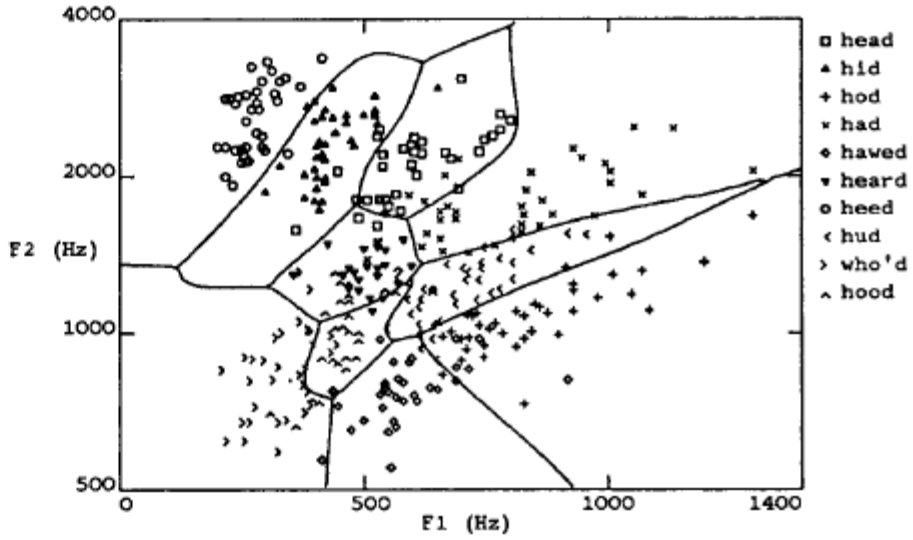


Multilayer Network

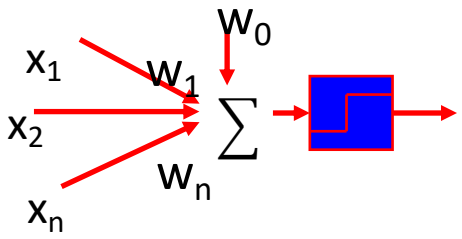
Solution

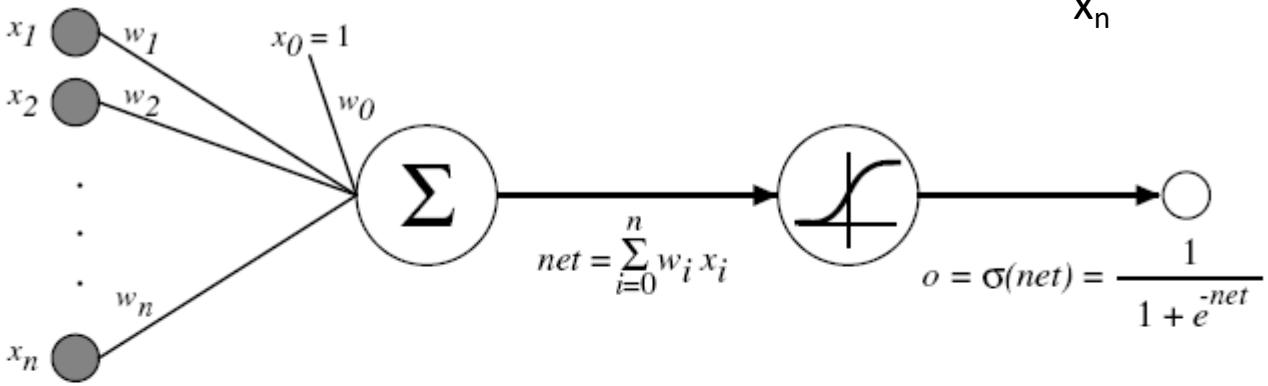
We can build a multilayer network represents the highly nonlinear decision surfaces.





Sigmoid Unit





$\sigma(x)$ is the sigmoid function

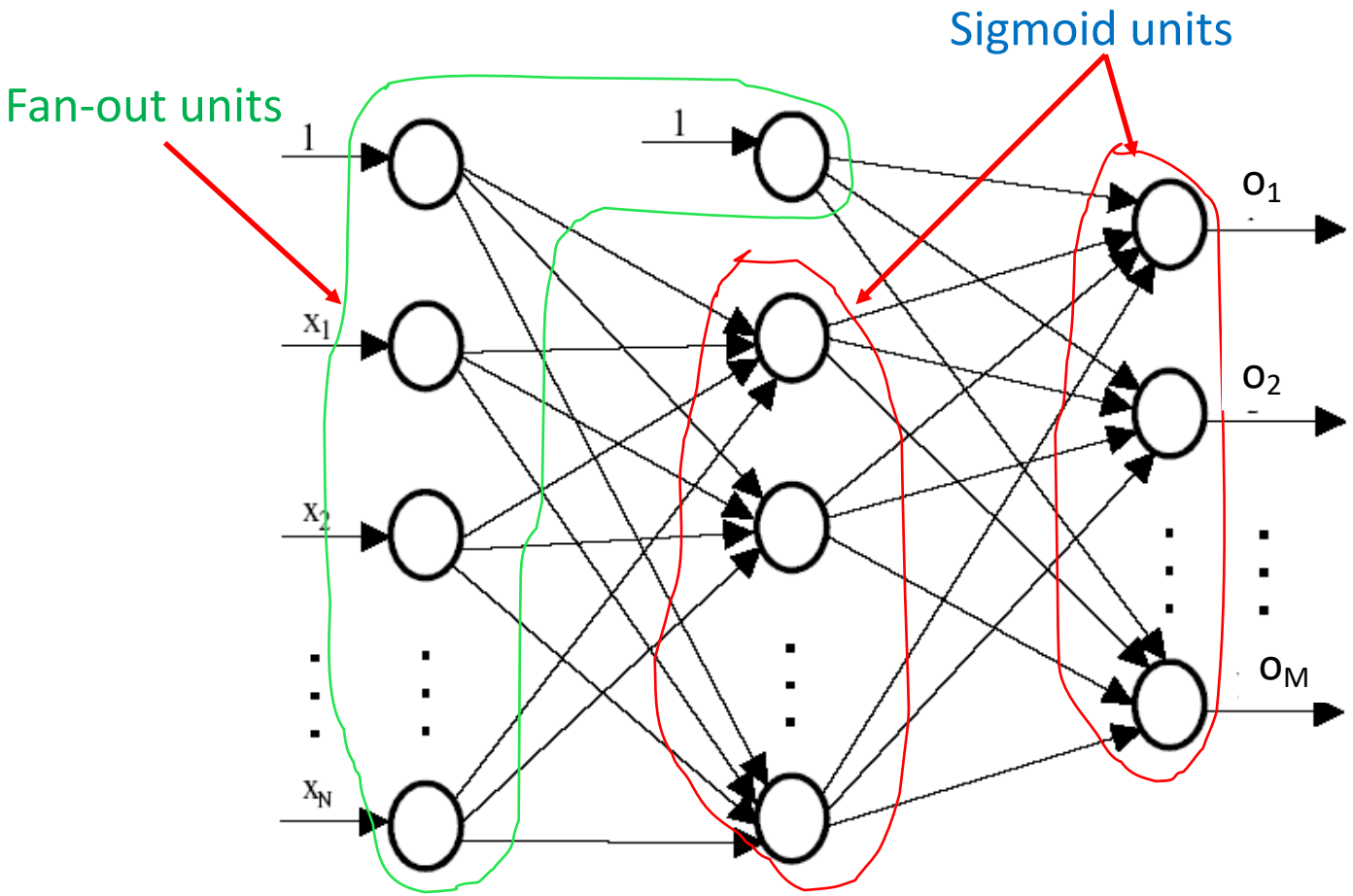
$$\frac{1}{1 + e^{-x}}$$

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

We can derive gradient decent rules to train

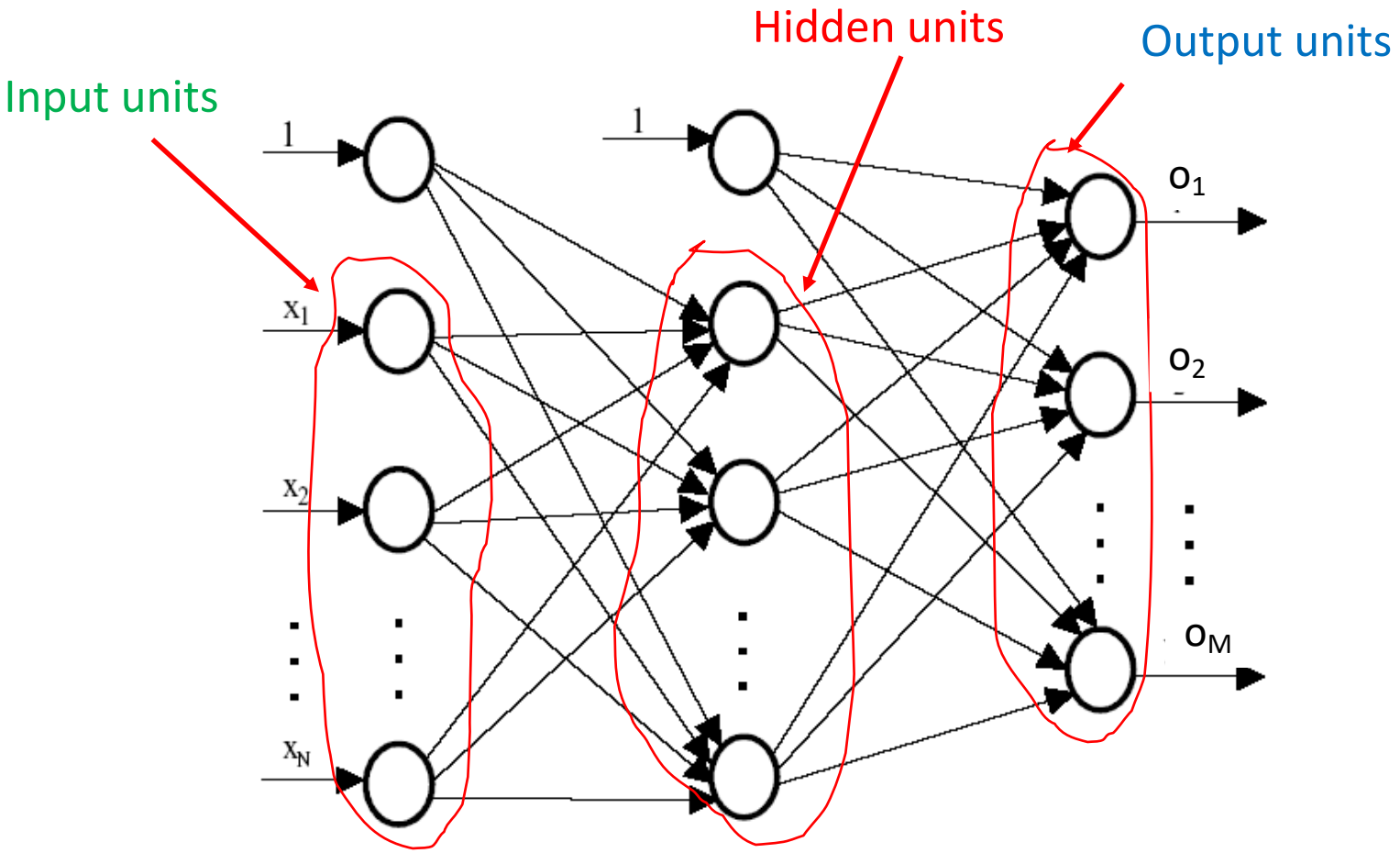
Multilayer Perceptron

A three layer perceptron

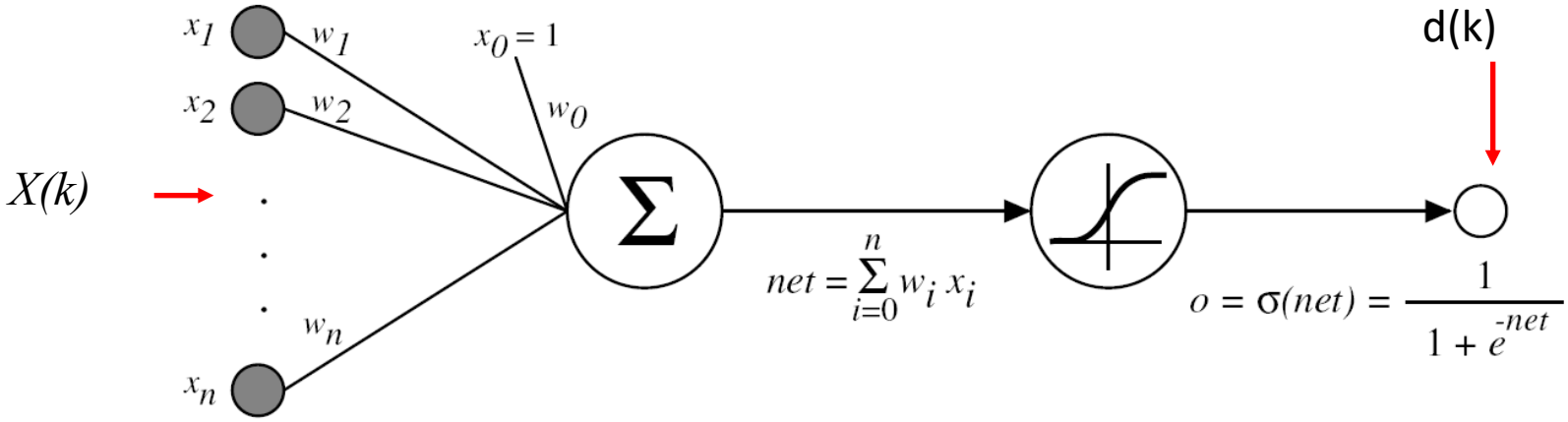


Multilayer Perceptron

A three layer perceptron



Error Gradient for a Sigmoid Unit



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

Error Gradient for a Sigmoid Unit

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

$$\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial w_i}} = \frac{\partial E}{\partial w_i}\left(\frac{1}{2}\sum_{k=1}^{K}(d(k)-o(k))^2\right)$$

$$= \frac{1}{2}\sum_{k=1}^{K}\left(\frac{\partial E}{\partial w_i}(d(k)-o(k))^2\right)$$

$$= \frac{1}{2}\sum_{k=1}^{K}\left(2(d(k)-o(k))\frac{\partial E}{\partial w_i}(d(k)-o(k))\right)$$

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

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

Error Gradient for a Sigmoid Unit

$$\frac{\partial o(k)}{\partial net(k)} = \frac{\partial \sigma(net(k))}{\partial net(k)} = \sigma\big(net(k)\big)\Big(1 - \sigma\big(net(k)\big)\Big) = o(k)(1 - o(k))$$

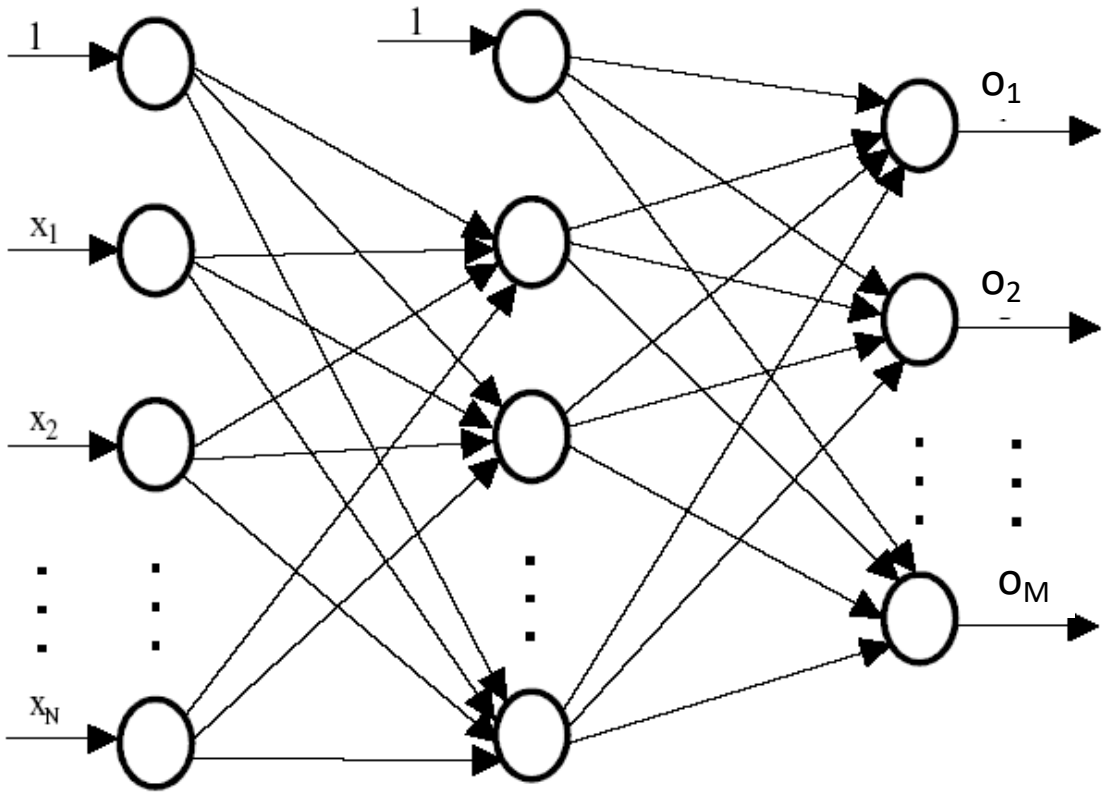
$$\frac{\partial net(k)}{\partial w_i} = \frac{\partial \sigma(WX(k))}{\partial w_i} = x_i(k)$$

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

$$= -\sum_{\boldsymbol{k=1}}^{\boldsymbol{K}}\Big(\big(\boldsymbol{d(k) - o(k)}\big)\boldsymbol{o(k)}\big(\boldsymbol{1 - o(k)}\big)\boldsymbol{x_i(k)}\Big)$$

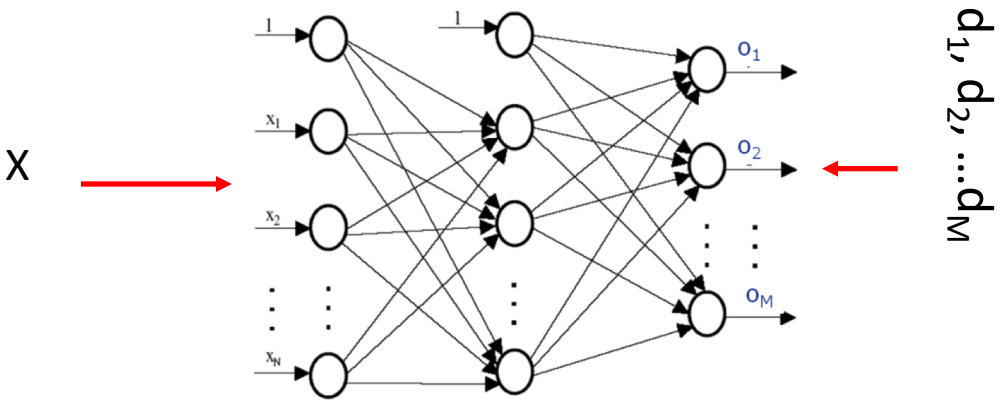
Back-propagation Algorithm

For training multilayer perceptrons



Back-propagation Algorithm

For each training example, training involves following steps

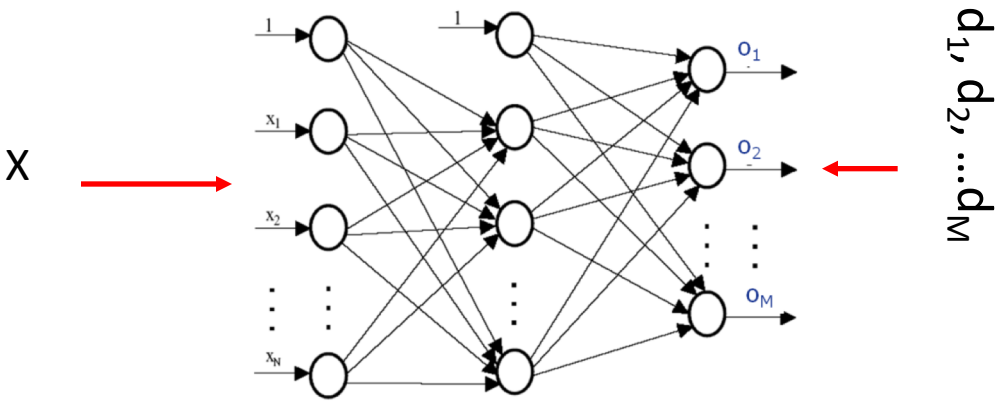


Step 1: Present the training sample, calculate the outputs

Note that we initialize all the weights similar to single perceptron.

Back-propagation Algorithm

For each training example, training involves following steps

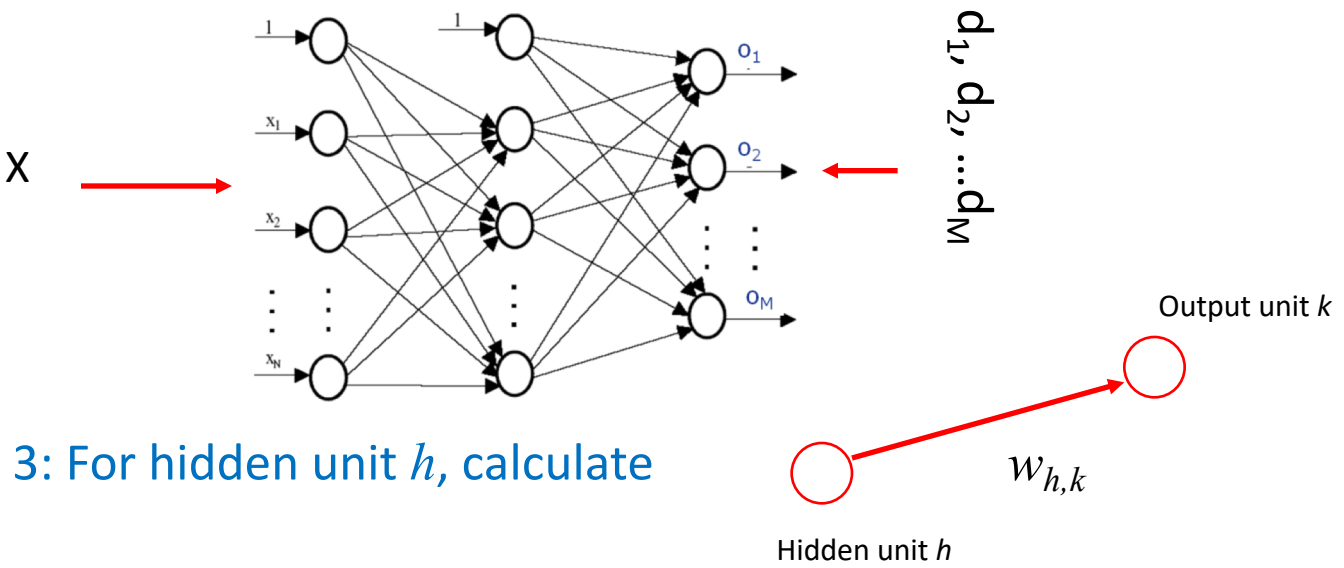


Step 2: For each output unit k, calculate

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

Back-propagation Algorithm

For each training example, training involves following steps



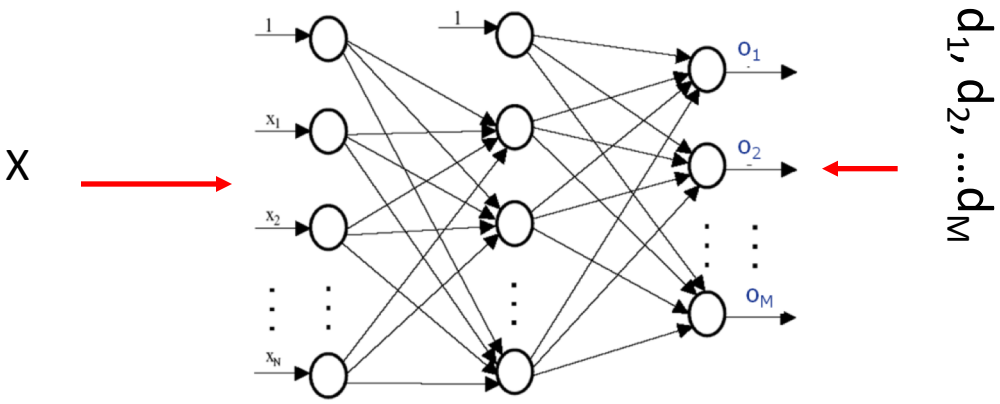
Step 3: For hidden unit h, calculate

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

Error back-propagation

Back-propagation Algorithm

For each training example, training involves following steps

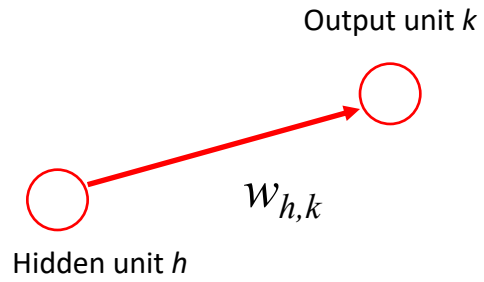


Step 4: Update the output layer weights, $w_{h,k}$

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

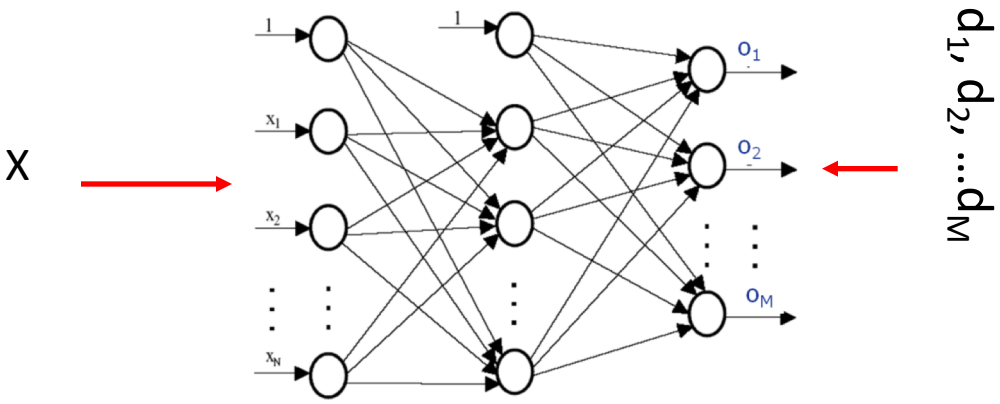
$$\Delta w_{h,k} = \eta \delta_k o_h$$

where o_h is the output of hidden layer h

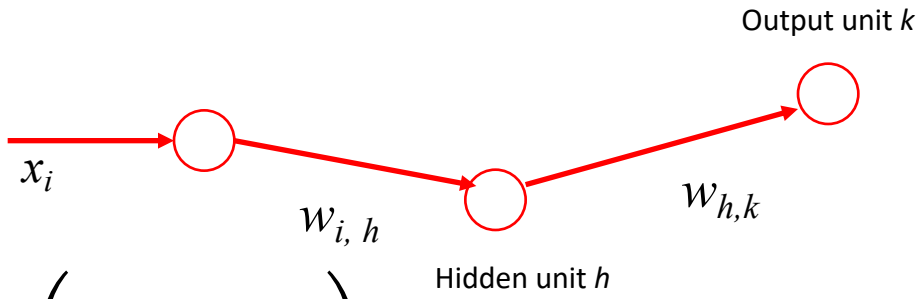


Back-propagation Algorithm

For each training example, training involves following steps



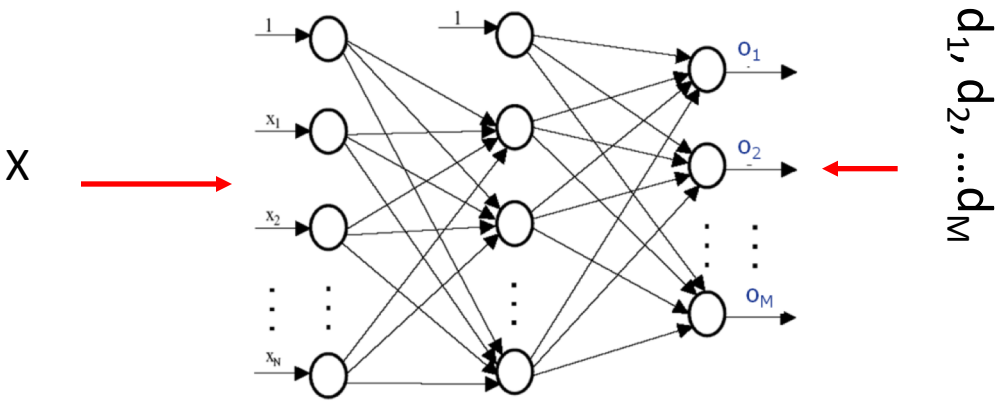
o_h is the output of hidden unit h



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

Back-propagation Algorithm

For each training example, training involves following steps



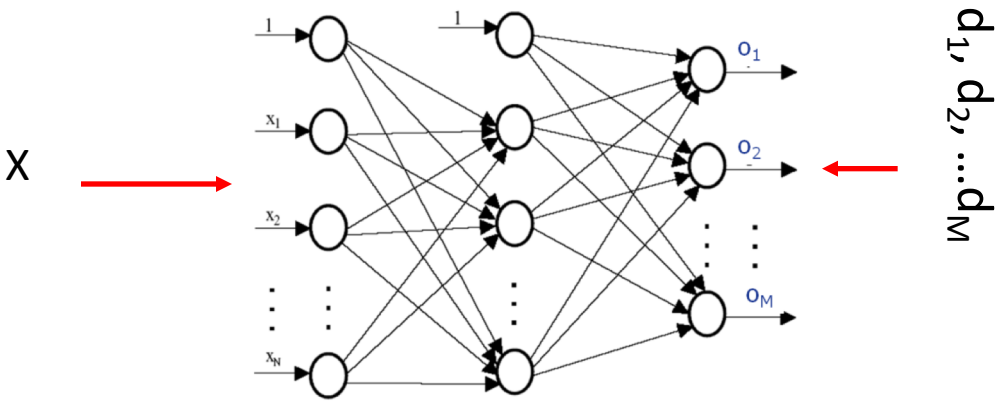
Step 4: Update the output layer weights, $w_{h,k}$

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

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

Back-propagation Algorithm

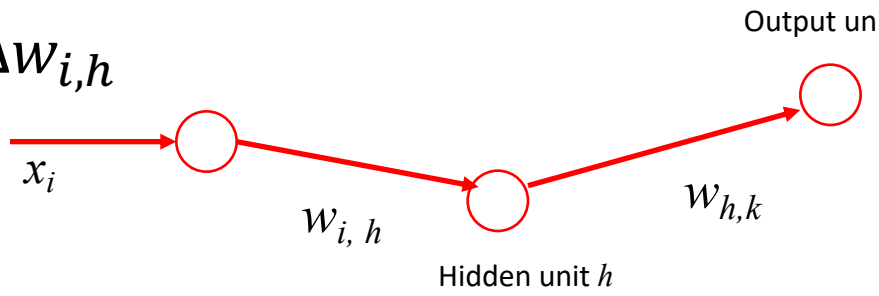
For each training example, training involves following steps



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

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

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



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 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.

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].

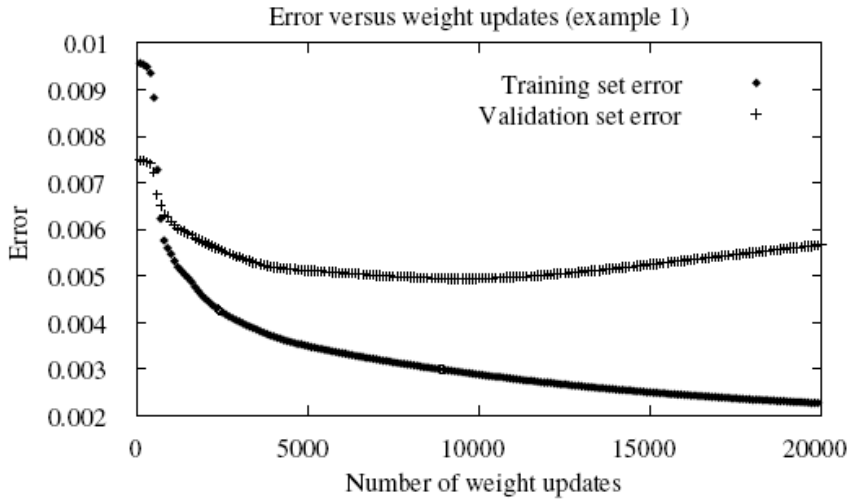
Generalization, Overfitting and Stopping Criterion

What is the appropriate condition for stopping weight update?

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.

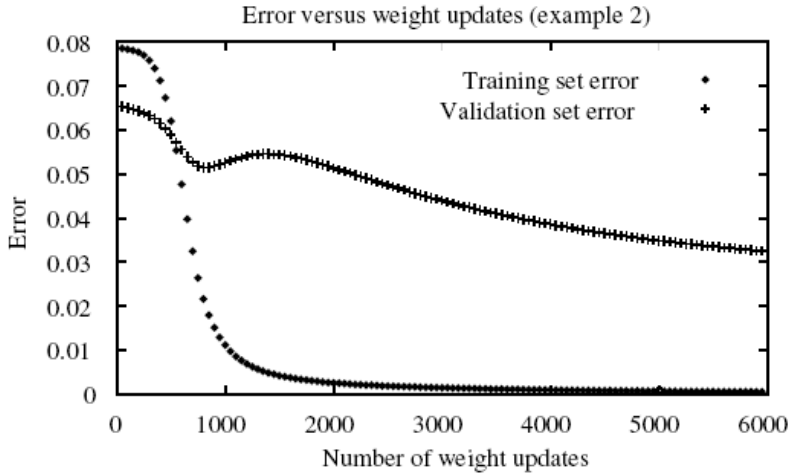
Generalization, Overfitting and Stopping Criterion



A training set

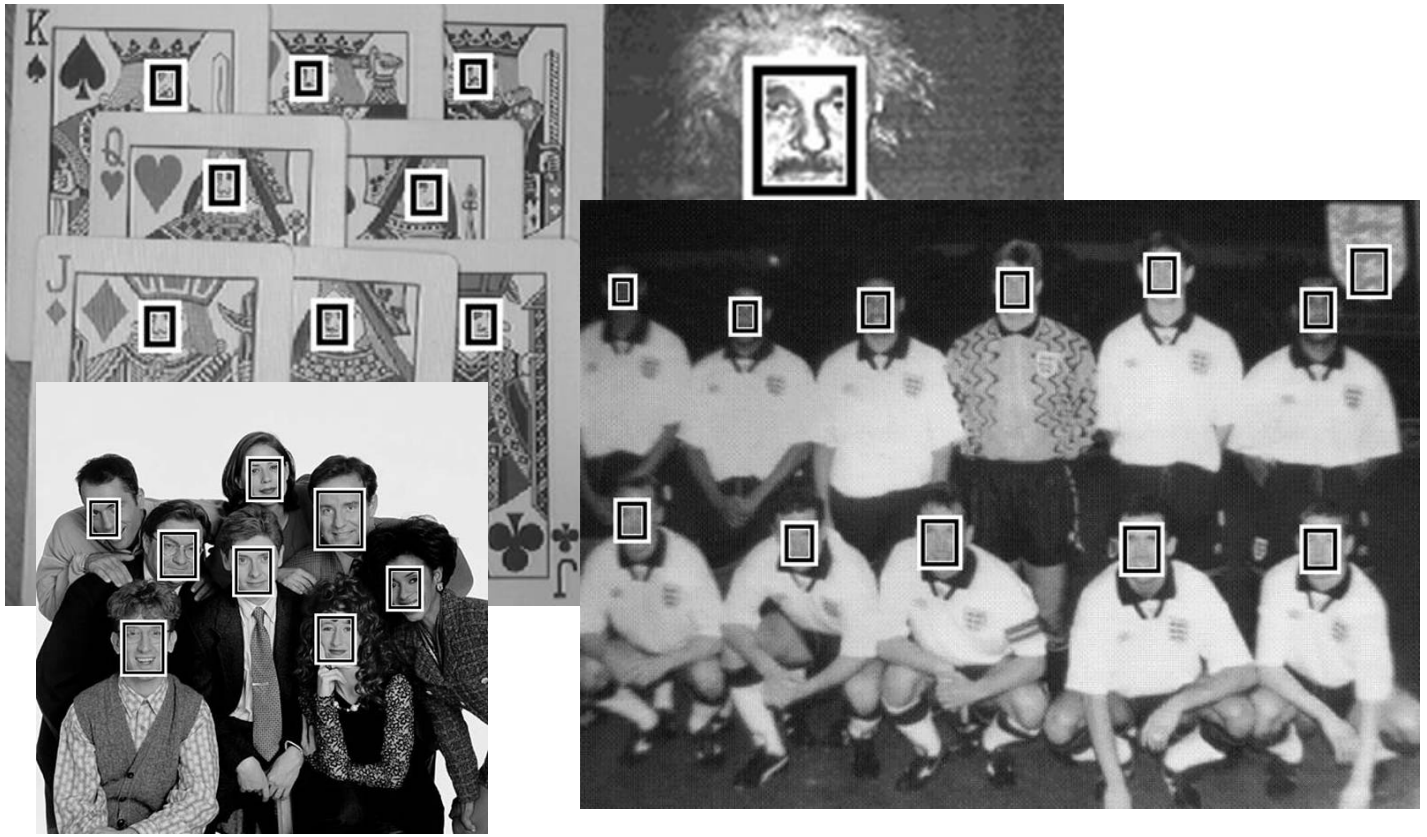
A validation set

Stop training when the validation set has the lowest error



Application Example

Neural Network-based Face Detection



https://ri.cmu.edu/pub_files/pub1/rowley_henry_1996_3/rowley_henry_1996_3.pdf

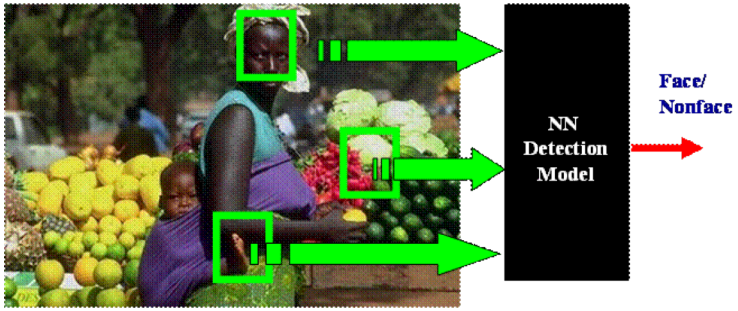
Application Example

Neural Network-based Face Detection

Application Example

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 Example

Neural Network-based Face Detection

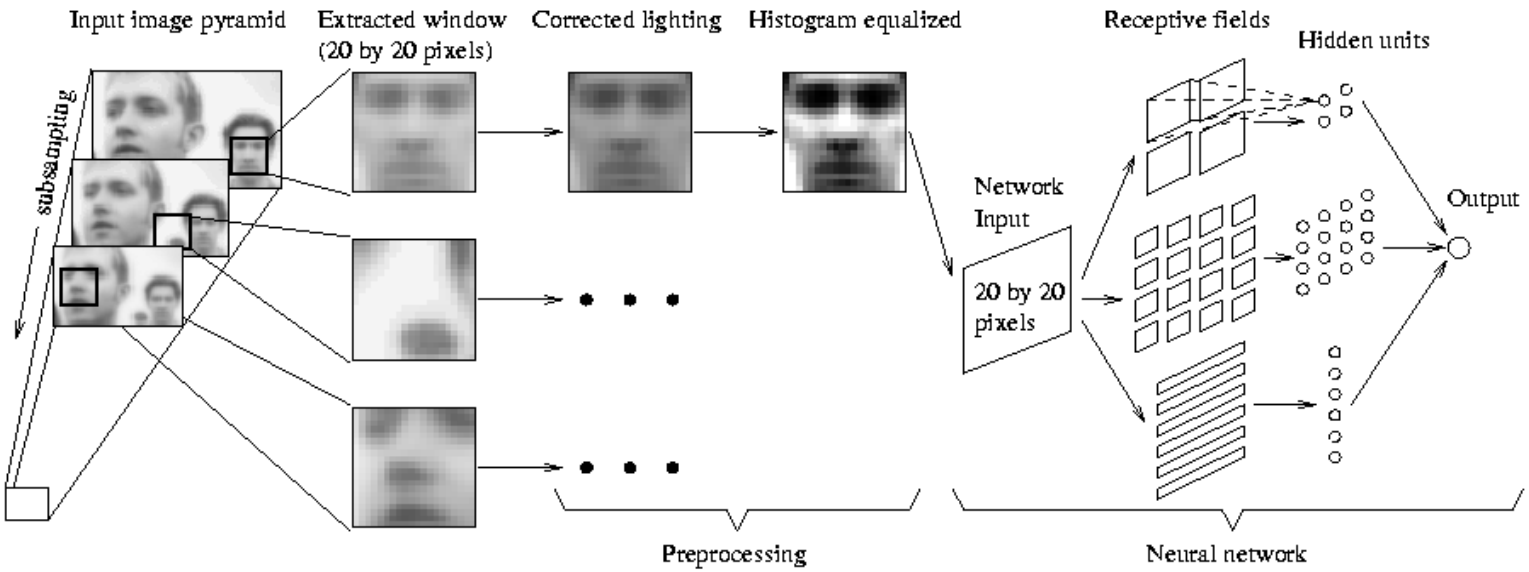


Figure 1: The basic algorithm used for face detection.

Application Example

Neural Network-based Face Detection

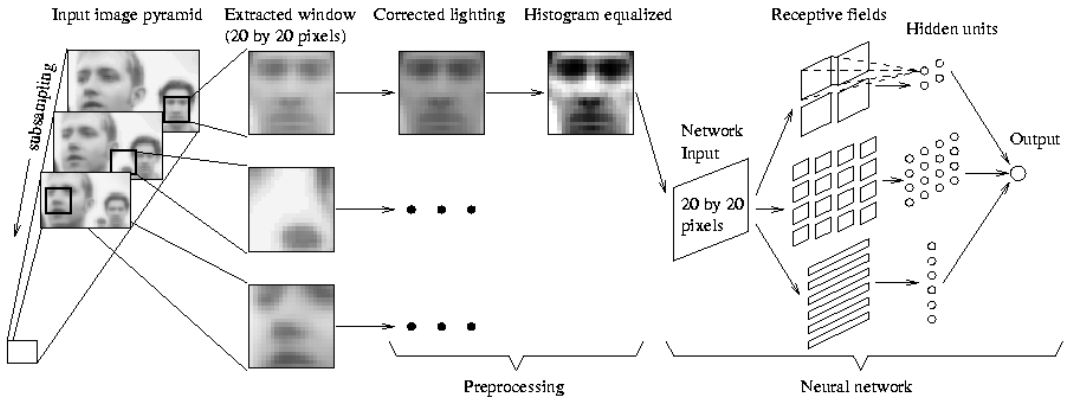


Figure 1: The basic algorithm used for face detection.

- 4 look at 10 x 10 subregions
- 16 look at 5x5 subregions
- 6 look at 20x5 horizontal stripes of pixels

Application Example

Neural Network-based Face Detection

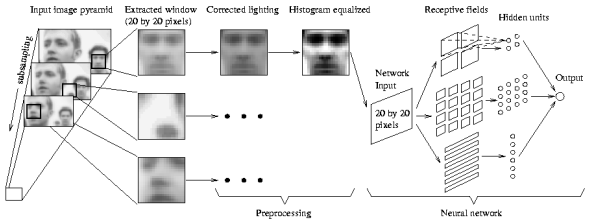


Figure 1: The basic algorithm used for face detection.

- 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 Example

Neural Network-based Face Detection

- Training samples: Non-face training samples

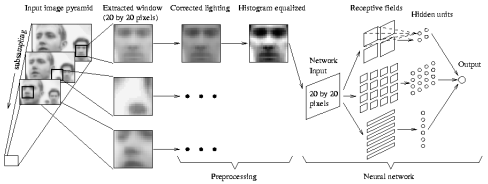
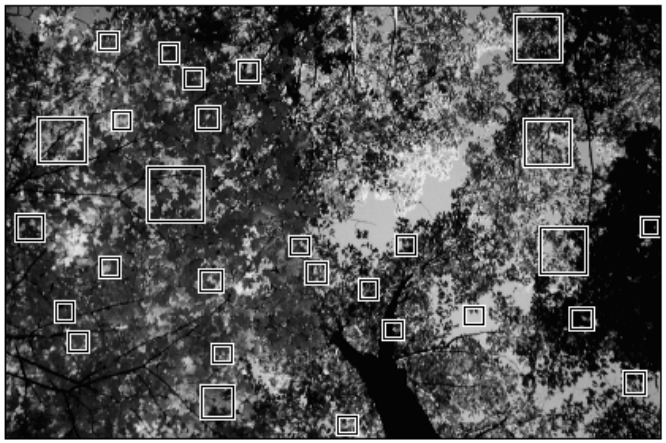
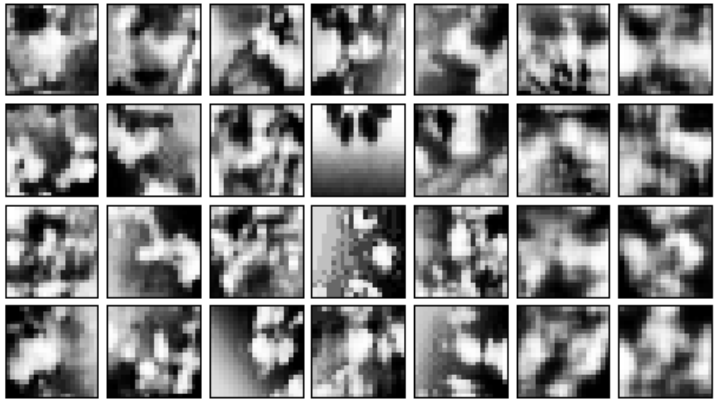


Figure 1: The basic algorithm used for face detection.





Application Example

Neural Network-based Face Detection

- Post-processing and face detection

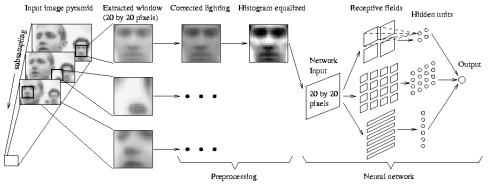
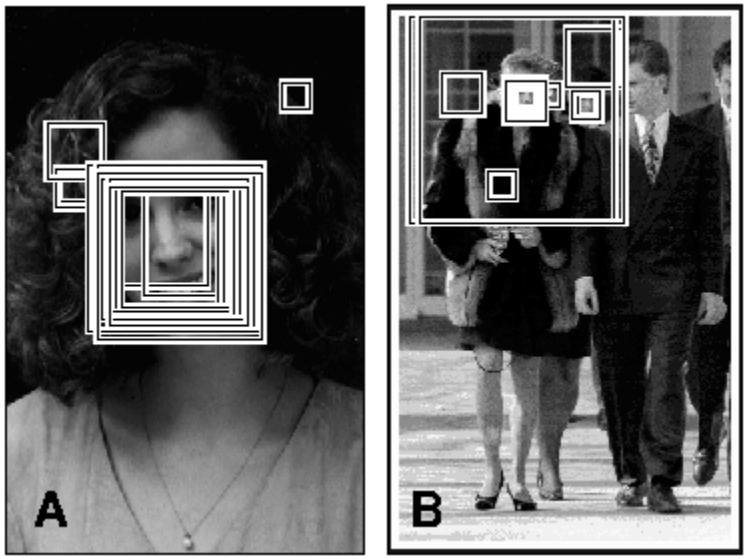


Figure 1: The basic algorithm used for face detection.



Application Examples

Neural Network-based Face Detection

- Results and Issues

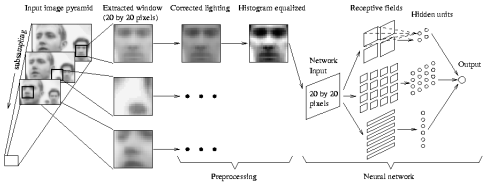


Figure 1: The basic algorithm used for face detection.

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

Further Reading

Chapter 4, T. M. Mitchell, Machine Learning, McGraw-Hill International Edition, 1997